CA-IDMS®

DML Reference — PL/I

15.0

Computer Associates™

Second Edition, October 2001

Contents

How to Use This Manual

What this manual is about

This document presents navigational and LRF DML statements for use in CA-IDMS/DB database and CA-IDMS/DC and CA-IDMS/UCF data communication environments.

Most *data communication* DML statements are applicable in both CA-IDMS/DC and CA-IDMS/UCF environments. The acronym DC/UCF is used to represent this.

Who should use this manual

This document is intended for use by PL/I programmers who run programs against CA-IDMS/DB databases and who want to use the DC/UCF system facilities.

What this manual contains

This manual contains six chapters and seven appendices:

- **Introduction to the CA-IDMS Data Manipulation Language** (Chapter 1)

  An overview of the facilities for preparing, compiling, and executing PL/I applications under CA-IDMS/DB and CA-IDMS/DC

- **DML Precompiler Options** (Chapter 2)

  A description of the precompiler options available in the database/data communications PL/I environment

- **Communications Blocks and Error Detection** (Chapter 3)

  A discussion of the communications blocks and error detection routines available in the database/data communications PL/I environment

- **Required PL/I Declaratives** (Chapter 4)

  Descriptions of the PL/I declarative statements required to program in the CA-IDMS/DB environment

- **DML Precompiler-Directive Statements** (Chapter 5)

  Instructions for using DML precompiler-directive statements

- **Data Manipulation Language Statements** (Chapter 6)

  Descriptions of the database and data communications PL/I DML commands, including syntax, resulting status codes, examples, and information on the impact of commands on currency

- **DML Precompile, PL/I Compile, and Link-Edit JCL** (Appendix A)

- **Call Formats** (Appendix B)

  Call-format expansions for PL/I DML statements

- **Keywords** (Appendix C)

  A list of CA-IDMS/DB and CA-IDMS/DC keywords and reserved words

- **Notes to Teleprocessing Monitor Users** (Appendix D)

  An outline of program specifications required to run CA-IDMS/DB and CA-IDMS/DC PL/I programs under CICS, INTERCOMM, SHADOW, and TASK/MASTER

- **Sample Programs and Database Definition** (Appendix E)

  - **Sample Batch Program**

    A sample PL/I program that retrieves database records by using DML statements

  - **Sample Online Program**

    A sample PL/I program that retrieves logical records by using Logical Record Facility DML statements

  - **EMPLOYEE Database Definition**

A data structure diagram

- **Considerations for IBM Language Environment** (Appendix F)

  An explanation of the IBM Language Environment and how to use it with CA-IDMS/DC

- **18-Byte Communications Blocks** (Appendix G)

  An explanation of the 18-byte IDMS-DB and IDMS-DC communications blocks

How to proceed

1. Read Chapter 1 for an overview of programming in the CA-IDMS/DB environment.
2. Use Chapters 2, 3, 4, and 5 for details relevant to PL/I programming in the CA-IDMS/DB environment.
3. Refer to syntax, syntax rules, and examples in Chapter 6, as needed, when coding DML statements.
4. Refer to supplemental information provided in the appendices, as needed.

Related documents

For more information related to this manual, refer to the following documents:

- *CA-IDMS Messages and Codes*
- CA-IDMS installation manual for your operating system
- *CA-IDMS System Generation*
- *CA-IDMS System Operations*
- *CA-IDMS Navigational DML Programming*
- *CA-IDMS Mapping Facility*
- *CA-IDMS Database Administration*
- *IDD DDDL Reference*

Understanding syntax diagrams

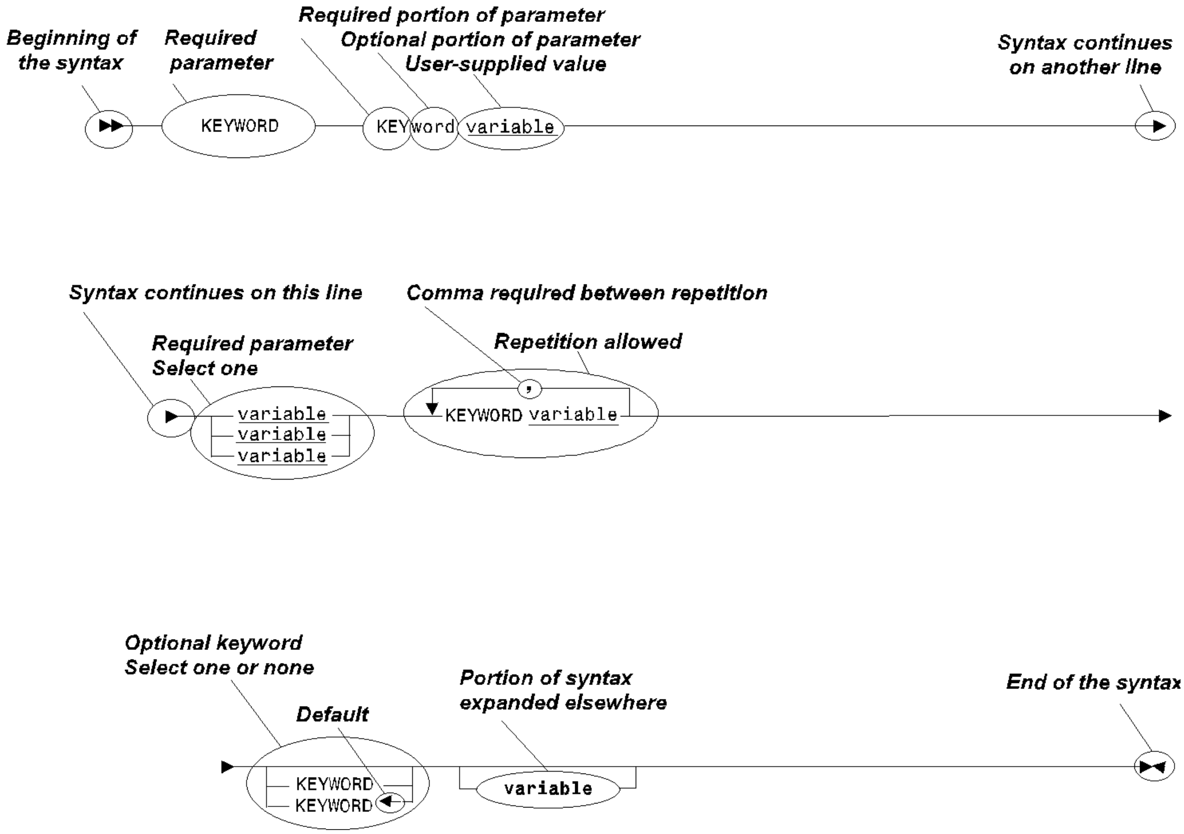
Look at the list of notation conventions below to see how syntax is presented in this manual. The example following the list shows how the conventions are used.

UPPERCASE OR SPECIAL CHARACTERS	Represents a required keyword, partial keyword, character, or symbol that must be entered completely as shown.
lowercase	Represents an optional keyword or partial keyword that, if used, must be entered completely as shown.
<u>underlined lowercase</u>	Represents a value that you supply.
←	Points to the default in a list of choices.
lowercase bold	Represents a portion of the syntax shown in greater detail at the end of the syntax or elsewhere in the document.
►►────────────────	Shows the beginning of a complete piece of syntax.
────────────────►◄	Shows the end of a complete piece of syntax.
────────────────►	Shows that the syntax continues on the next line.
►────────────────	Shows that the syntax continues on this line.
────────────────►─	Shows that the parameter continues on the next line.
─►───────────────	Shows that a parameter continues on this line.
►── parameter ──────►	Shows a required parameter.
►──┬─ parameter ─┬──► └─ parameter ─┘	Shows a choice of required parameters. You must select one.
►──┬─────────────┬──► └─ parameter ─┘	Shows an optional parameter.

►──┬─────────────┬──►
   ├─ parameter ─┤
   └─ parameter ─┘
	Shows a choice of optional parameters. Select one or none.

┌─────────────┐ ►─↓─ parameter ─┴──►	Shows that you can repeat the parameter or specify more than one parameter.
┌───── , ─────┐ ►─↓─ parameter ─┴──►	Shows that you must enter a comma between repetitions of the parameter.

Sample syntax diagram



Chapter 1. Introduction to CA-IDMS Data Manipulation Language

1.1 Overview

The CA-IDMS Data Manipulation Language (DML) consists of statements that direct CA-IDMS/DB database and data communications processing. You code DML statements in the program source as if they are a part of the host language. You use the DML PL/I compiler (also called the DMLP processor) to convert DML statements into standard PL/I statements. The DMLP processor also performs source-level error checking.

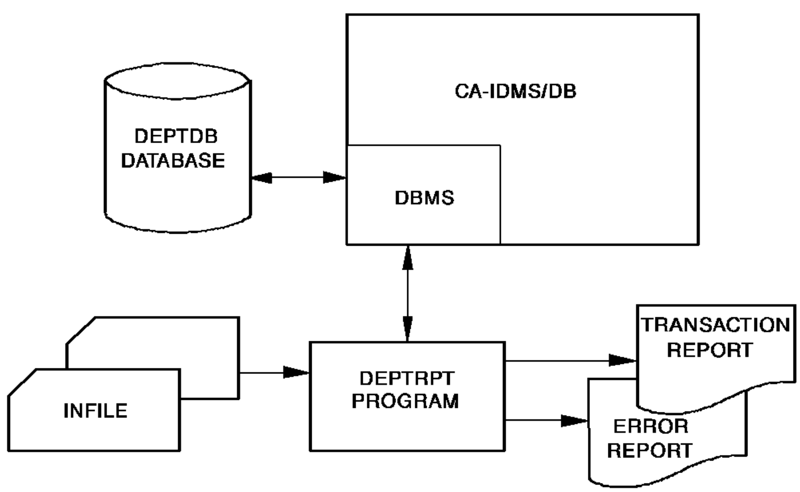
Your program uses different sets of DML statements, depending on whether its operating environment is batch or online. For example, a batch program uses only database DML statements. An online program uses data communications DML statements and can also use database DML statements.

1.1.1 Batch processing

Batch processing typically involves large volumes of transactions, sequential processing, and output in the form of files and reports. Batch programs use database DML statements only. Chapter 6, “Data Manipulation Language Statements” on page 6-1 contains all the DML commands, listed alphabetically. In this list, CA-IDMS/DC DML commands are distinguished from CA-IDMS/DB DML commands.

The following figure illustrates the flow of a typical batch application. Input to DEPTRPT consists of department IDs. Output consists of a listing of departments and their employees. The error report lists the department IDs of missing and empty departments.

Typical batch program flow

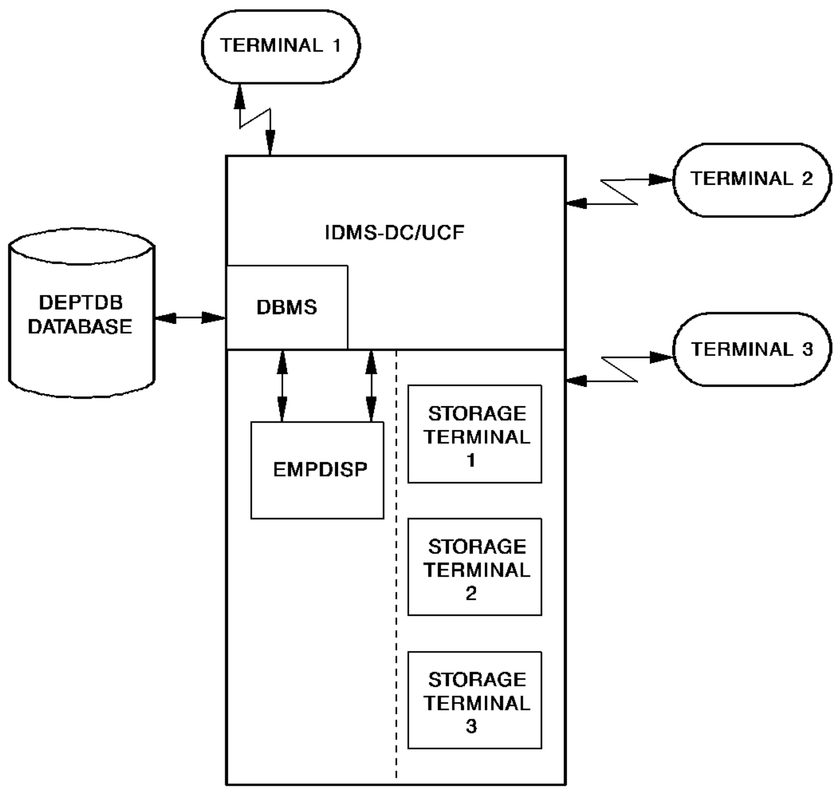


1.1.2 Online processing

Online processing typically involves transaction requests entered from terminals connected directly to the computer, transaction results displayed at the terminal, multiple requests from multiple sources, and sharing one copy of a program among multiple users. Additionally, online processing is immediate. The processing of large volumes of transactions from multiple online users requires fast response time. Online programs use data communications DML statements and can include database DML statements.

The following figure illustrates the flow of a typical online application. EMPDISP retrieves information for an operator-specified employee ID. Output to the terminal consists of DEPARTMENT, EMPLOYEE, JOB, and OFFICE information.

Typical online program flow



1.2 Programming in the CA-IDMS environment

CA-IDMS statements are either *database* or *data communications* statements. This section provides overview information and a more detailed subsection on each of the three types of database DML statements and on data communications statements.

Database statements: Database statements perform retrieval and update functions in either the batch or the online environment. These statements access database records and sets, one record at a time.

The three types of database statements are as follows:

- Navigational DML
- SQL DML
- Logical Record Facility DML

You can include database DML statements in batch programs or combine them with data communications DML statements in online programs that require database access.

Data communications statements: Data communications statements request data communications services, such as services for online programs.

Note: If you use a teleprocessing (TP) monitor other than CA-IDMS/DC, use CA-IDMS/DB DML statements only. Your TP monitor provides data communications services.

►► For information on accessing CA-IDMS/DC queues and printers from batch programs, see Chapter 5, “DML Precompiler-Directive Statements” on page 5-1.

1.2.1 Navigational DML

Navigational DML statements allow you to access database records and sets one record at a time, and to check and maintain currency in order to assure correct results. Navigational DML statements give you control over error checking and flexibility in choosing database access strategy. To use this type of DML statement, you must have a thorough knowledge of the database structure. For an example of a data structure diagram, see Appendix E, “Sample Programs and Database Definition” on page E-1.

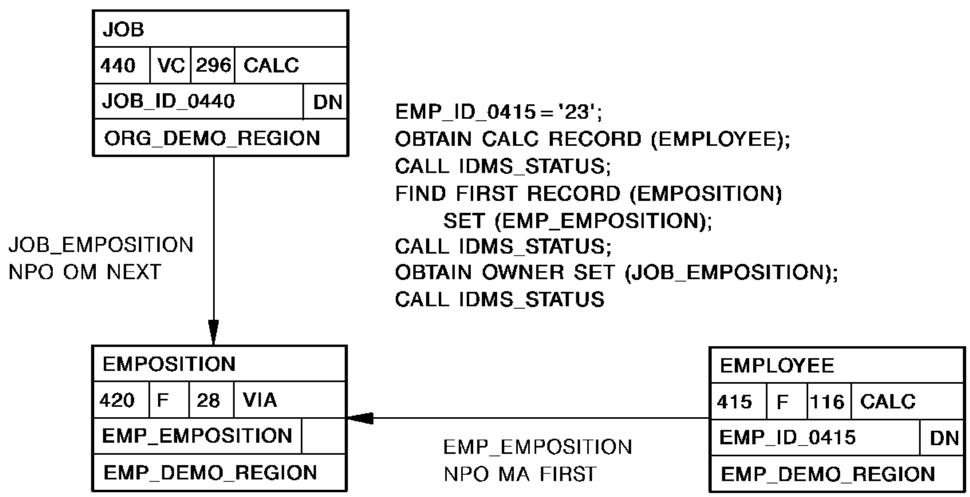
Navigational DML statements provide:

- **Control over error checking** You can check on the results of processing each statement.
- **Flexibility in choosing database access strategy** You can enter the database either sequentially (area sweep) by using a symbolic-key value (CALC or index), or by using a database-key value (DIRECT).

There are four types of navigational DML statements:

- **Control** statements initiate and terminate processing, effect recovery, prevent concurrent updates, and evaluate set conditions.
- **Retrieval** statements locate data in the database and make it available to the application program.
- **Modification** statements update the database.
- **Accept** statements pass database keys, storage address information, and statistics to the program.

Example of navigational DML statements: The following figure illustrates a database structure containing two owner records (EMPLOYEE and JOB) that share one member record (EMPOSITION), and lists the statements used to find employee and job information. To obtain EMPLOYEE and JOB information, you would retrieve an EMPLOYEE record, the first EMPOSITION record in the EMP_EMPOSITION set, and the owner record in the JOB_EMPOSITION set.



1.2.2 SQL DML

You can use SQL DML to access the same databases you access using navigational DML. Additionally, you can use SQL DML to access databases that have been defined using SQL DDL.

Using SQL DML, you do not have to be familiar with database structure and your programs do not have to include database navigation logic.

You can perform the following functions using SQL DML statements:

- Select rows
- Update rows
- Delete rows

- Insert rows

►► For more information about SQL DML statements, refer to the *CA-IDMS SQL Reference*. For information about embedding SQL statements in application programs, refer to *CA-IDMS SQL Programming*.

1.2.3 LRF DML

LRF (Logical Record Facility) statements allow you to access fields from multiple database records as if they are data fields in a single record. You specify selection criteria (using the WHERE clause) to access only the logical records you need.

Using LRF, you do not have to be familiar with database structure and your programs do not have to include database navigation logic.

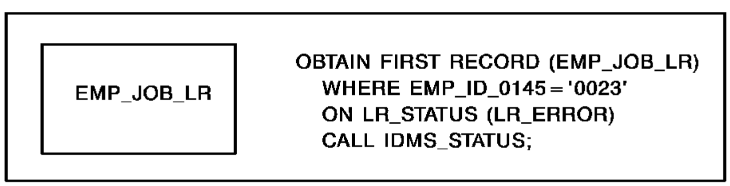
This manual describes these LRF DML statements:

- **ERASE** deletes a logical record as specified in the path definition
- **MODIFY** modifies a logical record as specified in the path definition
- **OBTAIN** retrieves a logical record as specified in the path definition
- **STORE** stores a logical record as specified in the path definition

Note: You must use the 48-character set for PL/I programs containing LRF DML (see 2.3, "PL/I compiler option usage" on page 2-5).

►► For information on path definition, refer to the *CA-IDMS Navigational DML Programming* document. For complete information on the Logical Record Facility, refer to the *CA-IDMS Logical Record Facility* document.

Example of LRF DML statements: The following figure illustrates the EMP_JOB_LR record. This record is a logical LRF record that contains the EMPLOYEE record, OFFICE record, and JOB record.



1.2.4 CA-IDMS/DC statements

CA-IDMS/DC and CA-IDMS/UCF are fully integrated with CA-IDMS/DB and the dictionary. They allow you to request both data communications and database services through standard subroutine calls generated (by the DML precompiler) from DML statements.

Example of a PL/I data stream with CA-IDMS/DC statements: The following is a typical PL/I data stream containing DML statements. The CA-IDMS/DC MAP IN, MAP OUT, and DC RETURN statements map in a user-specified employee ID, retrieve and display the specified information, and perform a DC RETURN naming TSK02 as the next task to be performed.

```
BIND MAP (EMPMAPLR);
BIND MAP (EMPMAPLR) RECORD (EMPLOYEE);
ACCEPT TASK CODE INTO (TASK_CODE_IN);
IF TASK_CODE_IN = 'TSK01' THEN
                GO TO INITIAL_MAPOUT;
MAP IN (EMPMAPLR);
 .
 .
 .
Database DML statements
 .
 .
 .
MAP OUT (EMPMAPLR)
        OUTPUT DATA YES
        MESSAGE (DISPLAY_MESSAGE) LENGTH (80);
DC RETURN NEXT TASK CODE ('TSK02');
```

Types of online CA-IDMS/DC statements: Online CA-IDMS/DC statements request that the DC/UCF system perform data communications services. There are nine types of online CA-IDMS/DC DML statements:

- **Program management** statements govern flow of control and abend processing.
- **Storage management** statements allocate and release variable storage.
- **Task management** statements provide runtime services that enhance control over task processing.
- **Time management** statements obtain the time and date, and define time-related events.
- **Scratch management** statements create, delete, or retrieve records from the scratch area.
- **Queue management** statements create, delete, or retrieve records in a queue area.
- **Terminal management** statements transfer data between the application program and a terminal.
- **Utility function** statements retrieve task-related information or statistics, send messages, and monitor access to database records.
- **Recovery** statements perform functions relating to database, scratch, and queue area recovery in the event of a system failure.

1.3 Compiling and executing programs

A PL/I program contains PL/I code and DML statements. The DML precompiler converts DML statements into PL/I CALL statements and copies information maintained in the dictionary into the source file. You can then compile, link edit, and execute the application program. The compilation and runtime processes are described separately below.

1.3.1 Compiling programs

These three components prepare a PL/I DML program for execution:

- The DML precompiler
- The PL/I compiler
- The linkage editor

Step 1 — DML precompiler

The DML precompiler converts DML statements: The DML precompiler converts DML statements in the source program to PL/I CALL statements and copies information maintained in the dictionary into the application program. For example, the DML precompiler could copy database record descriptions, map records, map definitions, and other predefined modules (such as communications blocks) into the program.

Output from the DML precompiler is a source file, which serves as input to the PL/I compiler, and an optional source listing. The output file differs from the source input to the DML precompiler in the following ways:

- Source code (such as the IDMS-DB communications block and the IDMS_STATUS routine) has been added to the program.
- DML statements have been replaced by PL/I CALL statements and changed to comment entries.

Additionally, the DML precompiler produces a listing of the following errors:

- Incorrect DML entries
- Statements inconsistent with the program's declared subschema view
- Any other error conditions detected during DMLP processing
- Warning messages indicating source code conditions that could adversely affect run units using the program

Step 2 — PL/I compiler

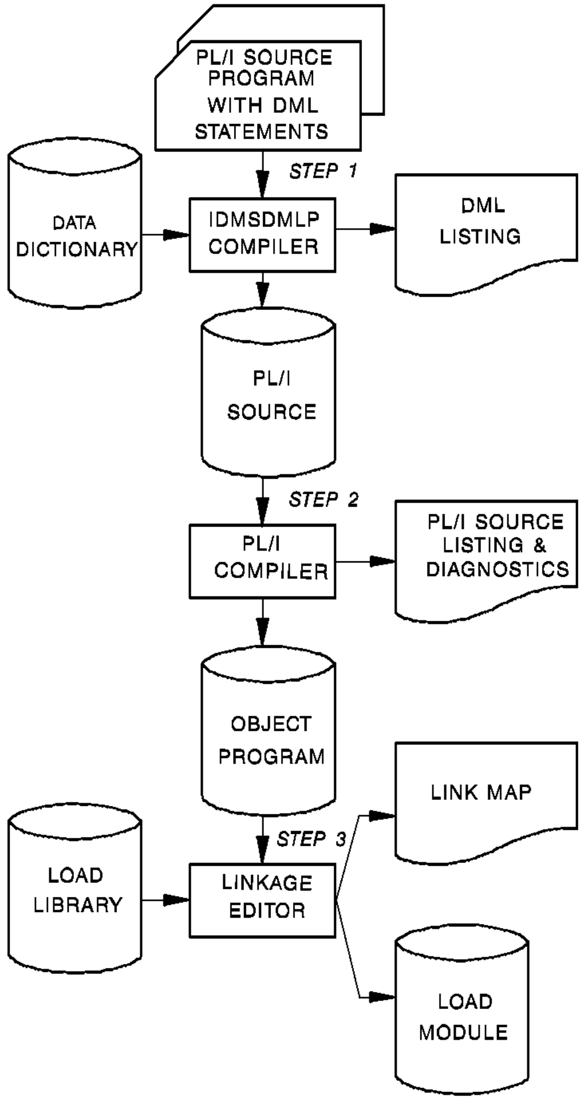
The PL/I compiler compiles the source into an object program: The PL/I compiler compiles the source program after the DML precompiler has processed it successfully. Output from the PL/I compiler consists of an object program and a source listing that includes any generated diagnostics.

Step 3 — Linkage editor

The linkage editor links the object program: The linkage editor link edits the object program into a specified load library. Output from the linkage editor consists of a load module (or phase) and a link map.

▸▸ See Appendix A, "DML Precompile, PL/I Compile, and Link-Edit JCL" on page A-1 for the job control language required to execute each step listed here.

PL/I program compile: The following figure illustrates a PL/I program compile.



1.3.2 Executing programs

At runtime, CA-IDMS requests are treated as application program subroutine calls. When an application program executes a CA-IDMS/DB or CA-IDMS/DC subroutine call, control passes to either CA-IDMS/DB or CA-IDMS/DC, which then processes the requested function.

A CA-IDMS/DC program must be defined to the CA-IDMS/DC system in which it will operate. The program can be defined either at system generation or at runtime by using a DCMT VARY DYNAMIC PROGRAM command.

►► For information on DCMT VARY DYNAMIC PROGRAM, refer to the *CA-IDMS System Tasks and Operator Commands* document.

PL/I features you cannot use: You *cannot* use the following PL/I features in programs running under CA-IDMS/DC:

- Any statement associated with file management: OPEN, CLOSE, DELETE, LOCATE, RELEASE, UNLOCK
- I/O statements: GET, READ, WRITE, REWRITE
- Any special feature that could generate a supervisor call (SVC): DATE, FETCH, DISPLAY, DELAY, WAIT, HALT, EVENT, COMPLETION, TIME, ATTN, ONCOUNT, ONKEY, ONFILE, ONSYSLOG
- The compile option: FLOW
- SPIE and STAE options (the DC/UCF system detects all runtime errors.)

Using these features inhibits system performance and can cause the DC/UCF system to abend.

Chapter 2. DML Precompiler Options

2.1 Overview

DML precompiler options are features of the DML programming environment that you select to customize the environment for your application program. This chapter describes these options and their associated syntax.

You code the DML precompiler options as special format entries in the PL/I source code input to the DML precompiler. Use the compiler options to:

- Override the DDLDML area default usage mode
- Enable the printing of dictionary and subschema comments
- Control the generation of DML precompiler source listings
- Suppress the logging of program activity statistics

This chapter provides a discussion of each of the compiler options.

2.2 Dictionary ready override

The DDLDML area is the main area of the dictionary accessed by the DML precompiler. Your application program can ready the DDLDML area using various usage mode options. The default mode is shared update usage. Shared update usage mode readies the DDLDML area for both retrieval and update. This mode also allows concurrently executing run units to ready the DDLDML area in shared update or shared retrieval usage mode. Your program can override the default usage mode by specifying either retrieval or protected update usage.

Syntax

```
►►─┬─ /*RETRIEVAL*/ ────────┬──────────────────────────────────────────►◄
   └─ /*PROTECTED_UPDATE*/ ─┘
```

```
Begin this syntax in column 2.
```

Parameters

RETRIEVAL
Readies the DDLDML area for retrieval only. It allows other concurrently executing run units to open the area in shared retrieval, shared update, protected retrieval, or protected update usage modes.

Note: If your program readies the DDLDML area for retrieval only, no program activity statistics can be logged.

PROTECTED_UPDATE
Readies the DDLDML area for both retrieval and update. It allows other concurrently executing run units to ready the area in retrieval usage mode only. The protected update usage mode prevents concurrent update of the area by run units executing in the same DC/UCF system.

Specify the dictionary ready override statement before all source input statements.

2.3 PL/I compiler option usage

The PROCESS statement is used to allow compile-time options to be specified for each compilation. For more information on these options, see a PL/I programming guide.

Syntax

```
►►── * PROCESS options; ──────────────────────────────────────────►◄

Begin this syntax in column 1.
```

If you use the PROCESS statement, it must follow the dictionary ready override statement. If you do not use the dictionary ready override statement, the PROCESS statement must precede all source input statements.

2.4 Comment generation

SCHEMA_COMMENTS generates the printing of the dictionary and subschema comments in a DML precompiler source listing.

Syntax

```
►►── /*SCHEMA_COMMENTS*/ ──────────────────────────────────────────►◄
```

```
Begin this syntax in column 2.
```

Code the SCHEMA_COMMENTS statement after the dictionary ready override and PROCESS CHARSET statements, if any, and before any source input statement.

Note: If you do not include the SCHEMA_COMMENTS statement in your source program, the DML precompiler does not generate comment lines.

2.5 List generation

The list generation option determines whether or not a DML source listing is generated.

You can turn source listing generation on or off any number of times in your source program. Do this by inserting appropriate NODMLIST/DMLIST entries in the code.

Note: DML always produces a listing of error messages. The DMLIST option controls output of the processor source listing only.

Syntax

```
►►──┬─ /*NODMLIST*/ ◄─ ─┬──────────────────────────────────────────────────────►◄
    └─ /*DMLIST*/ ──────┘
```

`Begin this syntax in column 2.`

Parameters

NODMLIST
Tells the DML precompiler not to generate the source listing for the statements that follow. NODMLIST is the default.

DMLIST
Tells the DML precompiler to generate the source listing for the statements that follow.

2.6 Log suppression

The NO_ACTIVITY_LOG option suppresses the logging of program activity statistics. The DML precompiler generates and logs the following program activity statistics unless you use the NO_ACTIVITY_LOG option:

- Program name
- Language
- Date last compiled
- Number of lines
- Number of compilations
- Date created
- Subschema name (if any)
- File statistics
- Database access statistics (records and modules copied from the dictionary; subprograms called; and records, sets, and areas accessed by DML verbs)

Note: Program activity statistics cannot be logged if you ready the DDLDML area for retrieval only or use a read-only dictionary. File activity statistics cannot be logged if you code both registered and unregistered program files in one OPEN statement.

Syntax

```
►►── /*NO_ACTIVITY_LOG*/ ─────────────────────────────────────────────►◄
```

```
Begin this syntax in column 2.
```

The NO_ACTIVITY_LOG statement must follow the dictionary ready override and PROCESS CHARSET statements, if any.

Chapter 3. Communications Blocks and Error Detection

3.1 Overview

This chapter describes the communications blocks available under CA-IDMS/DC and CA-IDMS/DB. These blocks return status information about requested database and data communications services to the application program. This chapter also describes the ERROR_STATUS field in the IDMS-DB and IDMS-DC communications blocks, error codes, and error detection routines.

3.2 Communications blocks

Communications blocks return status information about requested database (CA-IDMS/DB) and data communications (CA-IDMS/DC and CA-IDMS/UCF) services to the application program. Depending on the usage mode (LR, DML, or MIXED) defined in the subschema, your program uses one or two of the following blocks:

- **IDMS-DB communications block** — The IDMS-DB communications block is used when your program specifies the BATCH operating mode.
- **Logical-record request control (LRC) block** — The LRC block is used when the subschema usage mode is either LR or MIXED. The DML precompiler copies the LRC block with either the IDMS-DB communications block (operating mode of BATCH) or the IDMS-DC communications block (operating mode of IDMS_DC or DC_BATCH).
- **IDMS-DC communications block** — The IDMS-DC communications block is used when your program specifies either IDMS_DC or DC_BATCH operating mode.

Below, you will find a detailed discussion of these blocks. For more information on operating modes, see 5.2, “DECLARE SUBSCHEMA” on page 5-4.

3.2.1 IDMS-DB communications block

Your program uses the IDMS-DB communications block when the operating mode is BATCH. This communications block serves as an interface between the database management system (DBMS) and your application program. Whenever a run unit issues a call to the DBMS for a database operation, the DBMS returns information about the outcome of the requested service to your program's IDMS-DB communications block.

Your program instructs the DML precompiler to copy the data description (called SUBSCHEMA_CTRL) of the IDMS-DB communications block from the data dictionary into program variable storage. You accomplish this by coding an INCLUDE IDMS (SUBSCHEMA_CTRL) statement in your program. For further information on INCLUDE IDMS, see 5.4, “INCLUDE IDMS” on page 5-8.

You should examine the ERROR_STATUS field of the IDMS-DB communications block after every call to the DBMS. Depending on the value contained in this field, you should perform the IDMS_STATUS routine. For more information, see 3.3, “ERROR_STATUS field and codes” on page 3-17, later in this chapter. For example, if the ERROR_STATUS field contains the value 0307 while walking a set, your program should perform end-of-set processing. Otherwise, your program should perform the IDMS_STATUS routine.

Layout of the IDMS-DB communications block: The following figure shows the layout of the 16-byte IDMS-DB communications block. Note that the layout of the block differs for application programs running under CICS.

►► For information about the 18-byte IDMS-DB communications block, see Appendix G, "18-Byte Communications Blocks" on page G-1.

IDMS-DB 16-byte communications block

	Bytes	Field	Data Type	Length (bytes)	Suggested Initial Value
*	1 8	PROGRAM_NAME	Alphanumeric	8	Program Name
	9 12	ERROR_STATUS	Alphanumeric	4	'1400'
	13 16	DBKEY	Binary	4 (Fullword)	0000
	17 32	RECORD_NAME	Alphanumeric	16	Spaces
	33 48	AREA_NAME	Alphanumeric	16	Spaces
	49 64	ERROR_SET	Alphanumeric	16	Spaces
	65 80	ERROR_RECORD	Alphanumeric	16	Spaces
	81 96	ERROR_AREA	Alphanumeric	16	Spaces
**	97 100	PAGE_INFO	Binary	4 (Fullword)	0000
	97 ... 196	IDBMSCOM_AREA	Alphanumeric	100	Spaces
	197 200	DIRECT_DBKEY	Binary	4 (Fullword)	0000
NON-CICS	201 207	Reserved for system	Alphanumeric	7	Spaces
NON-CICS	208	FILLER	...	1	...
NON-CICS	209 212	RECORD_OCCUR	Binary	4 (Fullword)	0000
NON-CICS	213 216	DML_SEQUENCE	Binary	4 (Fullword)	0000
CICS	201 216	FILLER	...	16	Spaces
CICS	217 223	Reserved for system	Alphanumeric	7	Spaces
CICS	224	FILLER	...	1	...
CICS	225 228	RECORD_OCCUR	Binary	4 (Fullword)	0000
CICS	229 232	DML_SEQUENCE	Binary	4 (Fullword)	0000

```
 * word aligned
** PAGE_INFO_GROUP overlays bytes 97 and 98 and PAGE_INFO_DBK_FORMAT
   overlays bytes 99 and 100.  Both of these fields are binary datatype,
   each with a length of two bytes.  Suggested initial values for
   both are 00.  Together these two fields represent PAGE_INFO.
```

Fields containing program status information: The IDMS-DB fields shown in the following table contain program status information.

Field name	Description
PROGRAM_NAME	Alphanumeric field that contains the name of the program being executed. The DML precompiler initializes this field automatically, if the program contains an INCLUDE IDMS (SUBSCHEMA_BINDS) statement. If you do not include this statement in your program, you must initialize the field.
ERROR_STATUS	Alphanumeric field that contains a value indicating the outcome of the last DML statement executed. The DML precompiler initializes the ERROR_STATUS field to 1400. The DBMS updates this field after each database service request and before returning control to the program. The DBMS updates this field whether or not the request was processed successfully. For details on the ERROR_STATUS field and its use, see 3.3, “ERROR_STATUS field and codes” on page 3-17, later in this chapter. If your program consists of more than one run unit, it must reinitialize the ERROR_STATUS field to 1400 after finishing one run unit and before binding the next.
DBKEY	Binary fullword field that contains the database key of the last record accessed by the run unit. For example, after successful execution of a FIND command, the DBMS updates DBKEY with the database key of the located record. If the call to the DBMS results in an error condition, DBKEY remains unchanged.
RECORD_NAME	Alphanumeric field that contains the name of the last record successfully accessed by the run unit. This field is left justified and padded with spaces on the right.
AREA_NAME	Alphanumeric field that contains the name of the last area successfully accessed by the run unit. This field is left justified and padded with spaces on the right.
ERROR_SET	Alphanumeric field that contains the name of the set involved in the last operation that produced an error condition. This field is left justified and padded with spaces on the right.
ERROR_RECORD	Alphanumeric field that contains the name of the record involved in the last operation that produced an error condition. This field is left justified and padded with spaces on the right.

Field name	Description
ERROR_AREA	Alphanumeric field that contains the name of the area involved in the last operation that produced an error condition. This field is left justified and padded with spaces on the right.
IDBMSCOM_AREA	Alphanumeric field that is used internally by the DBMS for specification of runtime function information.

PAGE_INFO	Two binary halfwords that represent the page information associated with the last record accessed by the run unit. PAGE_INFO is not changed if the call to the DBMS results in a non-zero status. The first halfword (PAGE_INFO_GROUP) represents the page group number. The second halfword (PAGE_INFO_DBK_FORMAT) represents the db-key radix.
	
	The db-key radix portion of the page information can be used in interpreting a db-key for display purposes and in formatting a db-key from page and line numbers. The db-key radix represents the number of bits within a db-key value that are reserved for the line number of a record. By default, this value is 8, meaning that up to 255 records can be stored on a single page of the area. Given a db-key, you can separate its associated page number by dividing the db-key by 2 raised to the power of the db-key radix. For example, if the db-key radix is 4, you would divide the db-key value by 2**4. The resulting value is the page number of the db-key. To separate the line number, you would multiply the page number by 2 raised to the power of the db-key radix and subtract this value from the db-key value. The result would be the line number of the db-key. The following two formulas can be used to calculate the page and line numbers from a db-key value:
	`Page-number = db-key value / (2 ** db-key radix)`
	`Line-number = db-key value - (page-number * ( 2 ** db-key radix))`

DIRECT_DBKEY	Binary fullword field that contains either a db-key value that you specify or a null db-key value of -1. This field is used to store records with a location mode of DIRECT. Because the DBMS does not update this field, you must initialize DIRECT_DBKEY. This field can be used only when storing a record in a native VSAM relative record data set (RRDS). You must initialize DIRECT_DBKEY to the relative record number of the record being stored.

Fields for non-CICS application programs: The IDMS-DB fields in the following table should be used for non-CICS application programs.

Field name	Description
DATABASE_STATUS	Alphanumeric field reserved for use by the DBMS.
FILLER	Field used to ensure binary fullword alignment.
RECORD_OCCUR	Binary fullword field that contains a record occurrence sequence identifier used internally by the DBMS.
DML_SEQUENCE	Binary fullword field that contains the source-level sequence number generated by the DML precompiler. The DML precompiler updates this field before each call to the DBMS if you specify DEBUG in the DECLARE SUBSCHEMA statement. The runtime system does not use this field.

Fields for CICS application programs: The IDMS-DB fields in the following table should be used for CICS application programs.

Field name	Description
FILLER	Work area reserved for use by CICS applications.
DATABASE_STATUS	Alphanumeric field reserved for use by the DBMS.
FILLER	Field used to ensure binary fullword alignment.
RECORD_OCCUR	Binary fullword field that contains a record occurrence sequence identifier used internally by the DBMS.
DML_SEQUENCE	Binary fullword field that contains the source-level sequence number generated by the DML precompiler. The precompiler updates this field before each call to the DBMS if you specify DEBUG in the DECLARE SUBSCHEMA statement. The runtime system does not use this field.

Updating fields in the IDMS-DB communications block: After a call to the DBMS, one or more of these fields may have been updated, depending on the DML statement issued and whether the statement executed successfully.

Example of updated fields: The following figure illustrates the IDMS-DB communications block fields updated by successful and unsuccessful calls to the DBMS; only those fields accessed by the runtime system are shown.

Key for this figure:

*	If true, the field is set to zoned decimal zeroes (0000). If false, the field is set to 1601.
0	The field is set to zoned decimal zeroes.

Y	The field is updated.
C	The field is cleared to spaces.
N	The field is set to null db-key value (-1)
nn	Specific minor status code

	SUCCESSFUL										UNSUCCESSFUL									
	PROGRAM-NAME	ERROR-STATUS	DBKEY	RECORD-NAME	AREA-NAME	ERROR-SET	ERROR-RECORD	ERROR-AREA	PAGE-INFO	DIRECT-DBKEY	PROGRAM-NAME	ERROR-STATUS	DBKEY	RECORD-NAME	AREA-NAME	ERROR-SET	ERROR-RECORD	ERROR-AREA	PAGE-INFO	DIRECT-DBKEY
Control statements																				
BIND RUN-UNIT		0										14nn								
BIND RECORD		0										14nn				Y	Y	Y		
BIND PROCEDURE		0										14nn				Y	Y	Y		
READY		0										09nn				C	C	C		
FINISH		0	N	C		C	C	C				01nn				C	C	C		
COMMIT (ALL)		0	N	C		C	C	C				18nn				C	C	C		
ROLLBAK (CONTINUE)		0	N	C		C	C	C				19nn				C	C	C		
KEEP (EXCLUSIVE)		0	Y	Y	Y	C	C	C	Y			06nn				Y	Y	Y		
IF SET		*	Y	Y	Y	C	C	C	Y			16nn				Y	Y	Y		
IF NOT SET		*	Y	Y	Y	C	C	C	Y			16nn				Y	Y	Y		
Retrieval statements																				
FIND/OBTAIN RECORD		0	Y	Y	Y	C	C	C	Y			03nn				Y	Y	Y		
GET RECORD		0	Y	Y	Y	C	C	C	Y			05nn				Y	Y	Y		
RETURN RECORD		0	Y	Y	Y	C	C	C	Y			17nn				Y	Y	Y		
Modification statements																				
STORE RECORD		0	Y	Y	Y	C	C	C	Y			12nn				Y	Y	Y		
CONNECT RECORD		0	Y	Y	Y	C	C	C	Y			07nn				Y	Y	Y		
MODIFY RECORD		0	Y	Y	Y	C	C	C	Y			08nn				Y	Y	Y		
DISCONNECT RECORD		0	Y	Y	Y	C	C	C	Y			11nn				Y	Y	Y		
ERASE RECORD		0	N	Y	Y	C	C	C				02nn				Y	Y	Y		
Accept statements																				
ACCEPT DBKEY FROM CURRENCY		0				C	C	C				15nn				Y	Y	Y		
ACCEPT DBKEY REL TO CURRENCY		0				C	C	C				15nn				Y	Y	Y		
ACCEPT IDMS STATISTICS		0				C	C	C				15nn				Y	Y	Y		
ACCEPT BIND RECORD		0				C	C	C				15nn				Y	Y	Y		
ACCEPT PROCEDURE		0				C	C	C				82nn				Y	Y	Y		
ACCEPT PAGE_INFO		0				C	C	C				15nn				Y	Y	Y		

3.2.2 LRC block

Your program uses the logical-record request control (LRC) block when the subschema usage mode is LR or MIXED. The LRC block provides an interface between the Logical Record Facility (LRF) and the application program. It passes information about a logical-record request to LRF and returns path status information about the processing of the request to the program. You use the LRC block in conjunction with the IDMS-DB or IDMS-DC communications block.

Your program instructs the DML precompiler to copy the data description (called SUBSCHEMA_LR_CTRL) of the LRC block from the data dictionary into program variable storage. You accomplish this by coding an INCLUDE IDMS (SUBSCHEMA_LR_CTRL) statement in your program. For further information on INCLUDE IDMS, see 5.4, "INCLUDE IDMS" on page 5-8.

You should examine the LR_STATUS field of the LRC block after every call to LRF to determine the status of the call after processing. If the DBMS returns the value LR_ERROR, you should examine the ERROR_STATUS field of the IDMS-DB or IDMS-DC communications block.

Layout of the LRC block: The following figure shows the layout of the LRC block.

LRC BLOCK

	Field	Data Type	Length (bytes)	Suggested Initial Value
1 2	LRC-LRPXELNG	Binary	2 (Halfword)	00
3 4	LRC-MAXVXP	Binary	2 (Halfword)	00
5 8	LRIDENT	Alphanumeric	4	'LRC'
9 16	LRVERB	Alphanumeric	8	Spaces
17 32	LRNAME	Alphanumeric	16	Spaces
33 48	LR-STATUS	Alphanumeric	16	Spaces
49 64	FILLER	...	16	...
65 ... (variable-length)	PXE	Mixed	...	...

* word aligned

Description of fields: The LRC block contains the fields described in the following table.

Field name	Description
LRC_LRPXELNG	Specifies the length of the LRC block
LRC_MAXVXP	Specifies the length of the work area required to evaluate the WHERE clause.

Field name	Description
LRIDENT	Contains the constant LRC followed by a space.
LRVERB	Contains the verb passed to the Logical Record Facility.
LRNAME	Contains the name of the logical record being accessed.
LR_STATUS	Contains the path status of a logical-record request. Path statuses are 1- to 16-character strings; they can be either standard or defined in the subschema by the DBA. LRF provides three standard path statuses: LR_FOUND, LR_NOT_FOUND, and LR_ERROR. ►► For more information on path statuses, see 6.78, “Logical-record clauses (WHERE and ON)” on page 6-215.
FILLER	Work area used internally by the Logical Record Facility.
PXE (WHERE clause)	Contains the expansion of the WHERE clause; it can contain from 0 to 512 1-byte elements. The 512-byte limit can be raised or lowered by using the SIZE parameter of the INCLUDE IDMS (SUBSCHEMA_LR_CTRL) statement. ►► For more information on the SIZE parameter and the INCLUDE IDMS statement, see 5.4, “INCLUDE IDMS” on page 5-8.

3.2.3 IDMS-DC communications block

The IDMS-DC communications block replaces the IDMS-DB communications block when the operating mode is either IDMS_DC or DC_BATCH. At runtime, the DC/UCF system uses the IDMS-DC communications block to pass information about the outcome of requested data communications and database services to an application program.

Your program instructs the DML precompiler to copy the data description (called SUBSCHEMA_CTRL) of the IDMS-DC communications block from the dictionary into program variable storage. You accomplish this by coding an INCLUDE IDMS (SUBSCHEMA_CTRL) statement in your program. For further information on INCLUDE IDMS, see 5.4, “INCLUDE IDMS” on page 5-8.

You should examine the ERROR_STATUS field of the IDMS-DC communications block after every call to the DBMS. Depending on the value contained in this field, you should perform the IDMS_STATUS routine. For more information, see 3.3, “ERROR_STATUS field and codes” on page 3-17, later in this chapter.

Layout of the IDMS-DC communications block: The following figure shows the layout of the 16-byte IDMS-DC communications block.

►► For information about the 18-byte IDMS-DC communications block, see Appendix G, "18-Byte Communications Blocks" on page G-1.

16-byte IDMS-DC communications block

	Bytes	Field	Data Type	Length (bytes)	Suggested Initial Value
*	1 8	PROGRAM	Alphanumeric	8	Program Name
	9 12	ERROR_STATUS	Alphanumeric	4	'1400'
	13 16	DBKEY	Binary	4 (Fullword)	0000
	17 32	RECORD_NAME	Alphanumeric	16	Spaces
	33 48	AREA_NAME	Alphanumeric	16	Spaces
	49 64	ERROR_SET	Alphanumeric	16	Spaces
	65 80	ERROR_RECORD	Alphanumeric	16	Spaces
	81 96	ERROR_AREA	Alphanumeric	16	Spaces
**	97 100	PAGE_INFO	Binary	4 (Fullword)	0000
	97 ... 196	IDBMSCOM_AREA	Alphanumeric	100	Spaces
	197 200	DIRECT_DBKEY	Binary	4	0000
	201 ... 300	DCBMSCOM_AREA	Alphanumeric	100	Spaces
	301 304	SSC_ERRSTAT_SAVE	Alphanumeric	4	Spaces
	305 308	SSC_DMLSEQ_SAVE	Binary	4 (Fullword)	0000
	309 312	DML_SEQUENCE	Binary	4 (Fullword)	0000
	313 316	RECORD_OCCUR	Binary	4 (Fullword)	0000
	317 320	SUBSCHEMA_CTRL_END	Alphanumeric	4	Spaces

* word aligned
** PAGE_INFO_GROUP overlays bytes 97 and 98 and PAGE_INFO_DBK_FORMAT overlays bytes 99 and 100. Both of these fields are binary datatype each having a length of two bytes. Suggested initial values for both are 00. Together these two fields represent PAGE_INFO.

Description of fields: The IDMS-DC communications block contains the fields described in the following table.

Field name	Description
PROGRAM	Contains your application program's name. If you code an INCLUDE IDMS(SUBSCHEMA_BINDS) statement in your program, the DML precompiler initializes this field automatically. If you do not include this statement in your program, you must initialize the field.

Field name	Description
ERROR_STATUS	Contains a value indicating the outcome of the last DML statement executed. The DML precompiler initializes the ERROR_STATUS field to 1400. The DC/UCF system updates this field after a requested database or data communications service call and before returning control to your program. The DC/UCF system updates this field whether or not the request was processed successfully. If your program consists of more than one run unit, it must reinitialize the ERROR_STATUS field to 1400 after finishing one run unit and before binding to the next. ►► For details on the ERROR_STATUS field and its use, see 3.3, “ERROR_STATUS field and codes” on page 3-17, later in this chapter.
DBKEY	Contains the database key of the last record accessed by the run unit. For example, after successful execution of a FIND command, the DBMS updates DBKEY with the database key of the located record. If the database call results in an error condition, DBKEY remains unchanged.
RECORD_NAME	Contains the name of the last record accessed successfully by the run unit. This field is left justified and padded with spaces on the right.
AREA_NAME	Contains the name of the last area accessed successfully by the run unit. This field is left justified and padded with spaces on the right.
ERROR_SET	Contains the name of the set involved in the last operation to produce an error condition. This field is left justified and padded with spaces on the right.
ERROR_RECORD	Contains the name of the record involved in the last operation to produce an error condition. This field is left justified and padded with spaces on the right.
ERROR_AREA	Contains the name of the area involved in the last operation to produce an error condition. This field is left justified and padded with spaces on the right.
IDBMSCOM_AREA	Used internally by the DBMS for specification of runtime information.

Field name	Description

PAGE_INFO	Two binary halfwords that represent the page information associated with the last record accessed by the run unit. PAGE_INFO is not changed if the call to the DBMS results in a non-zero status. The first halfword (PAGE_INFO_GROUP) represents the page group number. The second halfword (PAGE_INFO_DBK_FORMAT) represents the db-key radix.
	
	The db-key radix portion of the page information can be used in interpreting a db-key for display purposes and in formatting a db-key from page and line numbers. The db-key radix represents the number of bits within a db-key value that are reserved for the line number of a record. By default, this value is 8, meaning that up to 255 records can be stored on a single page of the area. Given a db-key, you can separate its associated page number by dividing the db-key by 2 raised to the power of the db-key radix. For example, if the db-key radix is 4, you would divide the db-key value by 2**4. The resulting value is the page number of the db-key. To separate the line number, you would multiply the page number by 2 raised to the power of the db-key radix and subtract this value from the db-key value. The result would be the line number of the db-key. The following two formulas can be used to calculate the page and line numbers from a db-key value:
	`Page-number = db-key value / (2 ** db-key radix)`
	`Line-number = db-key value - (page-number * ( 2 ** db-key radix))`

DIRECT_DBKEY	Contains either a user-specified db-key value or a null db-key value of -1. This field is used to store records with a location mode of DIRECT. Because the DC/UCF does not update this field, you must initialize DIRECT_DBKEY. A note for native VSAM users: use the DIRECT_DBKEY field only when storing a record in a native VSAM relative record dataset (RRDS). You must initialize DIRECT_DBKEY to the relative record number of the record being stored.
DCBMSCOM_AREA	Used internally by the DC/UCF system for specification of runtime function information.
SSC_ERRSTAT_SAVE	Used by the IDMS_STATUS routine to save a nonzero ERROR_STATUS in the event of an abend.

Field name	Description
SSC_DMLSEQ_SAVE	Used by the IDMS_STATUS routine to save the value of DML_SEQUENCE in the event of an abend.
DML_SEQUENCE	Contains the source-level sequence number generated by the DML precompiler. The DML precompiler updates this field before each call to the system if you specify DEBUG in the DECLARE SUBSCHEMA statement. The runtime system does not use this field.
RECORD_OCCUR	Contains a record occurrence sequence identifier used internally by the system.
SUBSCHEMA_CTRL_END	Marks the end of the IDMS-DC communications block.

3.3 ERROR_STATUS field and codes

You can use the ERROR_STATUS field of the IDMS-DB or IDMS-DC communications block to determine whether a DML request was processed successfully. The DBMS or the DC/UCF system returns a value to the ERROR_STATUS field, indicating the result of each DML request. For more information on using the ERROR_STATUS field, see 3.4, "Error detection" on page 3-28 later in this chapter.

LRF users: Check the LR_STATUS field of the LRC block before checking the ERROR_STATUS field.

Major and minor codes: The ERROR_STATUS field is a zoned decimal (CHAR(4)) field consisting of four bytes. The first two bytes represent a major code. The second two bytes represent a minor code. Each major code identifies a database or data communications function. Each minor code describes the possible status of that function.

Values of codes: If a function requested by your program completes successfully, a value of 0000 is returned. If any other value is returned, this indicates that the function requested by your program may not have completed successfully. However, whether this is an error depends on the result you expected. For example, if you code a FIND CALC command, you should anticipate the return of the value 0326 (which indicates that a record was not found) after the DBMS processes the request. This allows you to trap the error and continue processing.

CA-IDMS/DB major codes range from 01 to 20. They occur during database access in either batch processing or online processing. DC/UCF major codes occur in either online or DC_BATCH processing. Codes with a major code of 00 apply to all DML functions. In the tables below, you will find information on database codes and DC/UCF codes.

3.3.1 Database status codes

Database major codes: The following table lists the database major codes and their meanings. For complete information on return codes, refer to *CA-IDMS Messages and Codes*.

Major code	Database function
00	Any DML statement
01	FINISH
02	ERASE
03	FIND/OBTAIN
05	GET

Major code	Database function
06	KEEP
07	CONNECT
08	MODIFY
09	READY
11	DISCONNECT
12	STORE
14	BIND
15	ACCEPT
16	IF
17	RETURN
18	COMMIT
19	ROLLBACK
20	Logical Record Facility (LRF) requests

Database minor codes: The following table lists the database minor codes and their meanings. For complete information on return codes, refer to *CA-IDMS Messages and Codes*.

Minor code	Status of the database function
00	Combined with a major code of 00, this code indicates successful completion of the DML operation. Combined with a nonzero major code, this code indicates that the DML operation was not completed successfully due to central version causes, such as time-outs and program checks.
01	An area has not been readied. When this code is combined with a major code of 16, an IF operation has resulted in a valid false condition.
02	Either the db-key used with a FIND/OBTAIN DBKEY statement or the direct db-key suggested for a STORE is not within the page range for the specified record name.
04	The occurrence count of a variably-occurring element has been specified as either less than zero or greater than the maximum number of occurrences defined in the control element.
05	The specified DML function would have violated a duplicates-not-allowed option for a CALC, sorted, or index set.
06	No currency has been established for the named record, set, or area.

Minor code	Status of the database function
07	Either the end of a set, area, or index has been reached, or the set is empty.
08	Either the specified record, set, procedure, or LR verb is not in the subschema, or the specified record is not a member of the set.
09	The area has been readied with an incorrect usage mode.
10	An existing access restriction or subschema usage prohibits execution of the specified DML function. For Logical Record Facility (LRF) users, the subschema in use allows access to database records only. Combined with a major code of 00, this code means that the program attempted to access a database record, but the subschema in use allows access to logical records only.
11	The record cannot be stored in the specified area due to insufficient space.
12	There is no db-key for the record to be stored. This is a system internal error and should be reported to your DBA.
13	A current record of run unit either has not been established or has been nullified by a previous ERASE statement.
14	The CONNECT statement cannot be executed because the requested record has been defined as a mandatory automatic member of the set.
15	The DISCONNECT statement cannot be executed because the requested record has been defined as a mandatory member of the set.
16	The record cannot be connected to a set of which it is already a member.
18	The record was not bound.
20	The current record is not the same type as the specified record name.
21	Not all areas being used were readied in the correct usage mode.
22	The record name specified is not currently a member of the set name specified.
23	The area name specified is either not in the subschema or not an extent area. Alternatively, the record name specified was not defined within the area name specified.
25	No currency was established for the named set.
26	Either no duplicates exist for the named record, or the record name cannot be found.
28	The run unit attempted to ready an area that was readied previously.

Minor code	Status of the database function
29	The run unit attempted to place a lock on a record that is already locked by another run unit. A deadlock results. Unless the run unit issued either a FIND/OBTAIN KEEP EXCLUSIVE or a KEEP EXCLUSIVE, the run unit was aborted.
30	An attempt was made to erase the owner record of a nonempty set.
31	The retrieval statement format conflicts with the record's location mode.
32	An attempt to retrieve a CALC/DUPLICATE record was unsuccessful. The value of the CALC field in variable storage is not equal to the value of the CALC control element in the current record of run unit.
33	At least one set in which the record participates was not included in the subschema.
40	The WHERE clause in an OBTAIN NEXT logical-record request is inconsistent with a previous OBTAIN FIRST or OBTAIN NEXT command for the same record. Previously specified criteria, such as a reference to a key field, were changed. A path status of LR_ERROR was returned to the LRC block.
41	The subschema contains no path that matches the WHERE clause in a logical-record request. A path status of LR_ERROR was returned to the LRC block.
42	An ON clause included in the path by the DBA specified return of the LR_ERROR path status to the LRC block. An error occurred while processing the Logical Record Facility (LRF) request.
43	A program check was recognized during evaluation of a WHERE clause. The program check indicates that a WHERE clause specified comparison of a packed decimal field to an unpacked nonnumeric data field. Alternatively, data in variable storage or a database record does not conform to its description. A path status of LR_ERROR was returned to the LRC block, unless the DBA included an ON clause to override this action in the path.
44	The WHERE clause in a logical-record request does not supply a key element (sort key, CALC key, or db-key) expected by the path. A path status of LR_ERROR is returned to the LRC block.
45	During evaluation of a WHERE clause, a program check was recognized because a subscript value is neither greater than 0 nor less than its maximum allowed value plus 1. A path status of LR_ERROR was returned to the LRC block, unless the DBA included an ON clause to override this action in the path.

Minor code	Status of the database function
46	A program check revealed an arithmetic exception (for example, overflow, underflow, significance, divide) during evaluation of a WHERE clause. A path status of LR_ERROR was returned to the LRC block, unless the DBA included an ON clause to override this action in the path.
53	The subschema definition of an indexed set does not match the indexed set's physical structure in the database.
54	Either the prefix length of an SR51 record is less than zero or the data length is less than or equal to zero.
55	An invalid length was defined for a variable-length record.
56	There is insufficient memory to accommodate the CA-IDMS/DB compression/decompression routines.
57	A retrieval-only run unit has detected an inconsistency in an index that should cause an 1143 abend, but optional APAR bit 216 has been turned on.
60	A record-occurrence type is inconsistent with the set named in the ERROR_SET field in the IDMS-DB communications block. This code usually indicates a broken chain.
61	No record can be found for an internal db-key. This code usually indicates a broken chain.
62	A system-generated db-key points to a record occurrence, but no record with that db-key can be found. This code usually indicates a broken chain.
63	The DBMS cannot interpret the DML function to be performed. When combined with a major code of 00, this code means that invalid function parameters were passed on the call to the DBMS. For LRF users, a WHERE clause includes a keyword that is longer than the 32 characters allowed.
64	The record cannot be found. The CALC-control element was not defined properly in the subschema.
65	The database page read was not the page requested.
66	The area specified is not available for update.
67	The subschema invoked does not match the subschema object tables.
68	The CICS interface was not started.
69	A BIND RUN_UNIT may not have been issued. The DC/UCF system may be inactive or may not be accepting new run units. Or the connection with the DC/UCF system may have been broken due to timeout or other factors. When combined with a major code of 00, this code means the program was disconnected from the DBMS.

Minor code	Status of the database function
70	The database or journal file will not ready properly. A JCL error is the probable cause.
71	The page range or page group for the area being readied, or the page requested cannot be found in the DMCL.
72	There is insufficient memory to load a subschema or database procedure dynamically.
73	A run unit, executing under the central version, will exceed the MAXERUS value specified at system generation.
74	The dynamic loading of a module failed. If the module ran under the central version, the DBMS did not find a subschema or database procedure module in the data dictionary or the load (core-image) library. Alternatively, if the DBMS loads the module, it will exceed the number of subschema and database procedures provided for at system generation.
75	A read error occurred.
76	A write error occurred.
77	A run unit has not been bound or has been bound twice. When combined with a major code of 00, this code means either the program is no longer signed on to the subschema or the variable subschema tables have been overwritten.
78	An area-wait deadlock occurred.
79	The run unit requested more db-key locks than are available to the system.
80	The target node is either not active or was disabled from the configuration.
81	The DC/UCF system does not know the database name specified.
82	The subschema is not valid under the specified database.
83	An error occurred in accessing native VSAM data sets.
87	The owner and member records for a set to be updated are not in the same page group or do not have the same db-key radix.

3.3.2 Data communications status codes

Data communications major codes: The following table lists data communications major codes and their meanings. For complete information on return codes, refer to *CA-IDMS Messages and Codes*.

Major code	Data communications function
00	Any DML statement
30	TRANSFER CONTROL
31	WAIT/POST
32	GET STORAGE/FREE STORAGE
33	SET ABEND EXIT/ABEND CODE
34	LOAD/DELETE TABLE
35	GET TIME/SET TIMER
36	WRITE LOG
37	ATTACH/CHANGE PRIORITY
38	BIND/ACCEPT/END TRANSACTION STATISTICS
39	ENQUEUE/DEQUEUE
40	SNAP
43	PUT/GET/DELETE SCRATCH
44	PUT/GET/DELETE QUEUE
45	BASIC MODE TERMINAL MANAGEMENT
46	MAPPING MODE TERMINAL MANAGEMENT
47	LINE MODE TERMINAL MANAGEMENT
48	ACCEPT/WRITE PRINTER
49	SEND MESSAGE
50	COMMIT TASK/ROLLBACK TASK/FINISH TASK/WRITE JOURNAL
51	KEEP LONGTERM

Data communications minor codes: The following table lists the data communications minor codes and their meanings. For complete information on return codes, refer to *CA-IDMS Messages and Codes*.

Minor code	Status of the data communications function
00	Combined with a major code of 00, this code indicates either that the DML function completed successfully or that all tested resources were enqueued.
01	The requested function cannot be performed immediately. Waiting will cause a deadlock.

Minor code	Status of the data communications function
02	Either the storage pool has insufficient storage or the storage required for control blocks is unavailable.
03	The scratch area ID cannot be found.
04	The queue ID (header) cannot be found. Alternatively, a paging session was in progress when the system received a second STARTPAGE command. An implied ENDPAGE was processed before the current STARTPAGE executed successfully.
05	The specified scratch record ID or queue record cannot be found.
06	No resource control element (RCE) exists for the queue record. Currency was not established.
07	Either an I/O error occurred during processing or the queue upper limit was reached.
08	The requested resource is not available.
09	The requested resource is available.
10	New storage was assigned.
11	A maximum task condition exists.
12	The named task code is invalid.
13	The named resource cannot be found.
14	A requested module, defined as nonconcurrent, is currently in use.
15	The named module was overlaid and cannot be reloaded immediately.
16	The specified interval control element (ICE) address cannot be found.
17	The record was replaced.
18	No printer terminals are defined for the current DC/UCF system.
19	The return area is too small. Data was truncated.
20	An I/O, program-not-found, or potential-deadlock error condition exists.
21	The message destination is undefined, the longterm id cannot be found, or a KEEP LONGTERM was issued by a nonterminal task.
22	The specified scratch area already contains a record.
23	No storage or resource control element (RCE) could be allocated for the reply area.
24	The maximum number of outstanding replies was exceeded.
25	An attention interrupt was received.

Minor code	Status of the data communications function
26	The output data stream contains a logical error.
27	A permanent I/O error occurred.
28	The terminal dial-up line is disconnected.
29	An invalid parameter was passed in the list set up by the DML precompiler.
30	The named function is not implemented.
31	An invalid parameter was passed. Alternatively, the TRB, LRB, or MRB contains an invalid field, or the request is invalid because of a possible logic error in the application program. In a DC_BATCH environment, the record length specified by the command probably exceeds the maximum length based on the packet size.
32	The derived length of the specified variable storage is either negative or zero.
33	Either the named table or the named map cannot be found in the dictionary load area.
35	A GET STORAGE request is invalid because the area in program variable storage was previously allocated.
36	The program was not defined during system generation or is marked out of service.
37	A GET STORAGE operand is invalid because the specified variable storage area is not based storage.
38	Either no GET STORAGE operand was specified or the specified variable was not previously allocated.
39	The terminal device requested is out of service.
40	NOIO was specified, but the data stream cannot be found.
41	An IF operation resulted in a valid true condition.
42	The named map does not support the online terminal device in use.
43	The terminal operator canceled a line I/O session.
44	The referenced field does not participate in the specified map. Your program may contain an invalid subscript.
45	An invalid terminal type is associated with the issuing task.
46	A terminal I/O error occurred.
47	The named area has not been readied.
48	The run unit has not been bound.
49	NOWAIT was specified, but WAIT is required.

Minor code	Status of the data communications function
50	Statistics are not being kept.
51	A lock manager error occurred during processing of the request.
52	The specified table is missing or invalid.
53	A user-written edit routine resulted in an error.
54	Either internal data is invalid or a data conversion error occurred.
55	A user-written edit routine cannot be found.
56	No DFLDS were defined for the map.
57	The ID cannot be found, the ID is not longterm permanent, or another run unit is using the ID.
58	Either the LRID cannot be found or the maximum number of concurrent task threads was exceeded.
59	An error occurred in transferring the KEEP LONGTERM request to IDMSKEEP.
60	The requested KEEP LONGTERM lock id is already in use with a different page group.
61	The requested KEEP LONMGTERM lock id is already in use with a different DBKey format.
63	Invalid function parameters were passed on the call to the DBMS.
64	No detail exists for an update, therefore no action was taken. Alternatively, the requested node for a header or detail is either not present or not updated.
68	No more updated details exist to map in. Alternatively, the amount of storage defined for pageable maps at system generation is insufficient. In the latter case, the DBMS ignores subsequent MAP OUT DETAIL statements.
69	A DC_BATCH task requires more buffer space than was requested in the BIND TASK statement.
72	No detail occurrence, footer, or header fields exist to be mapped out by a MAP OUT RESUME command. Alternatively, the scratch record that contains the requested detail could not be accessed. The latter case is a mapping internal error; you should report it to the DBA.
76	The first screen page was transmitted to the terminal.
77	Either the program is no longer signed on to the subschema or the variable subschema tables were overwritten.

Minor code	Status of the data communications function
80	The target node is either not active or is disabled from the configuration. Alternatively, code 80 can signify that a complete map page was built.
96	There are too many active run units for the internal table.
97	DBIO/DBMS issued an invalid status. Check the system log file for details.
98	An unsupported PL/I compiler option (for example, DEBUG) may have been specified.
99	An unexpected internal return code was received. The terminal device is out of service.

3.4 Error detection

You must check the value returned to the ERROR_STATUS field after each DML request.

Note: If you are using the Logical Record Facility, you should check the LR_STATUS field of the LRC block before checking the ERROR_STATUS field.

IDMS_STATUS routine: CA-IDMS/DB, CA-IDMS/DC, and CA-IDMS/UCF provide the IDMS_STATUS routine to help you find errors. IDMS_STATUS is an error-checking routine included in the dictionary. You can copy IDMS_STATUS into your program by coding the INCLUDE IDMS MODULE statement:

```
INCLUDE IDMS (IDMS_STATUS);
```

For more information on this statement, see 5.6, "INCLUDE IDMS MODULE" on page 5-16.

IDMS_STATUS routine used under batch: The following is the code that the INCLUDE IDMS (IDMS_STATUS) statement copies into batch programs.

```
IDMS_STATUS:  PROC;
     IF  ERROR_STATUS='0000' THEN GOTO END_STATUS;
     PUT SKIP EDIT ('PROGRAM NAME ------', PROGRAM,
                    'ERROR STATUS ------', ERROR_STATUS,
                    'ERROR RECORD ------', ERROR_RECORD,
                    'ERROR SET ---------', ERROR_SET,
                    'ERROR AREA --------', ERROR_AREA,
                    'LAST GOOD RECORD --', RECORD_NAME,
                    'LAST GOOD AREA ----', AREA_NAME,
                    'DML SEQUENCE ------', DML_SEQUENCE)
                   (A(19),X(5),A(8),SKIP,A(19),X(5),A(4),5(SKIP,
                   A(19),X(5),A(16)),SKIP,A(19),X(5),F(10));
     ROLLBACK;
     CALL ABORT;
END_STATUS: END;
```

IDMS_STATUS routine used under a DC/UCF system: The following is the code that the INCLUDE IDMS (IDMS_STATUS) statement copies into DC/UCF programs:

```
IDMS_STATUS:   PROC;
   IF ERROR_STATUS='0000' THEN GOTO END_STATUS;
   SSC_ERRSTAT_SAVE=ERROR_STATUS;
   SSC_DMLSEQ_SAVE=DML_SEQUENCE;
   SNAP FROM (SUBSCHEMA_CTRL) TO (SUBSCHEMA_CTRL_END);
   ABEND CODE (SSC_ERRSTAT_SAVE);
END_STATUS: END;
```

IDMS_STATUS abends your program if the ERROR_STATUS field contains a nonzero value. Since some values do not indicate processing errors, your program should check ERROR_STATUS for nonzero values before calling IDMS_STATUS.

Common status codes: The following table lists the most common codes to check before calling or executing IDMS_STATUS.

Status code	Description
0307	End of set, area, or index
0326	No record found
3101 3201 3401 3901	Waiting will cause a deadlock
3202 3204	Insufficient space available
4303	ID cannot be found
4404	Queue header cannot be found
4305 4404	Record cannot be found
3908	Resource not available
3909	Resource is available
3210	New space allocated
3711	Maximum attached tasks
4317	Record has been replaced
4319 4419 4519 4719	Return area too small; data has been truncated
4525 4625	Attention interrupt received
4743	The DC/UCF session was canceled by the operator

Status codes for pageable maps: The following table lists the status codes returned when you use pageable maps.

Status code	Description
4604	Second consecutive STARTPAGE
4664	No current detail
4668	All updated details mapped in or pageable map space exceeded
4672	Nothing to map out
4676	First page transmitted
4680	A complete map page was built

When IDMS_STATUS executes, it exits immediately if the error-status check indicates that the function completed successfully (error-status of 0000).

3.5 The effects of non-zero status on IDMS_STATUS

The following describes the effects of nonzero status conditions on IDMS_STATUS execution. The effects depend on the application program's operating mode (BATCH or IDMS_DC).

Effect when the operating mode is BATCH: When the operating mode is BATCH, a nonzero error status causes IDMS_STATUS to:

- Print status information on the unsuccessful function
- Issue a rollback
- Abend the program

The status information retrieved from the IDMS-DB communications block includes program name, error status, error record, error set, error area, record name (the last record successfully accessed), area name (the last area successfully accessed), and the DML sequence number.

Effect when the operating mode is IDMS_DC: When the operating mode is IDMS_DC, a nonzero error status causes IDMS_STATUS to:

- Snap the IDMS-DC communications block (SUBSCHEMA_CTRL)
- Abend the program

The status information retrieved from the IDMS-DC communications block includes program name, error status, error record, error set, error area, record name (the last record successfully accessed), area name (the last area successfully accessed), and the DML sequence number.

Chapter 4. Required PL/I Declaratives

4.1 Overview

This chapter describes the following PL/I declarative statements:

- DECLARE IDMS (for BATCH mode)
- DECLARE IDMSPLI (for IDMS_DC mode)
- DECLARE IDMSDCP (for DC_BATCH mode)
- DECLARE SQLXQ1 (for embedded SQL DML statements)
- DECLARE ADDR BUILTIN
- DECLARE ABORT
- DECLARE IDMSP

Note: For non-reentrant PL/I programs compiled under Release 2.3 of PL/I or earlier, you must specify OPTIONS (MAIN) in the PL/I PROCEDURE statement for the entry procedure. For reentrant PL/I Release 2.3 or earlier programs, you must specify OPTIONS (MAIN,REENTRANT). For AD/CYCLE (LE-COMPLIANT) PL/I programs, you must specify OPTIONS (REENTRANT,FETCHABLE).

4.2 DECLARE IDMS

Include the IDMS ENTRY statement for applications executing in BATCH mode.

```
►►─┬─ DECLARE ─┬─ IDMS ENTRY OPTIONS (INTER, ASSEMBLER); ──────────────────►◄
   └─ DCL ─────┘
```

4.3 DECLARE IDMSPLI

Include the IDMSPLI ENTRY statement for online applications executing in IDMS_DC mode.

```
►►─┬─ DECLARE ─┬─ IDMSPLI ENTRY OPTIONS (INTER, ASSEMBLER); ─────────────►◄
   └─ DCL ─────┘
```

4.4 DECLARE IDMSDCP

Include the IDMSDCP ENTRY statement for applications executing in DC_BATCH mode.

```
►►─┬─ DECLARE ─┬─ IDMSDCP ENTRY OPTIONS (INTER, ASSEMBLER); ──────────────────►◄
   └─ DCL ─────┘
```

4.5 DECLARE SQLXQ1

Include the SQLXQ1 ENTRY statement for applications with embedded SQL DML statements.

```
►►─┬─ DECLARE ─┬─ SQLXQ1 ENTRY OPTIONS (INTER, ASSEMBLER); ──────────────────►◄
   └─ DCL ─────┘
```

4.6 DECLARE ADDR BUILTIN

Include the ADDR BUILTIN statement so that all database and online application programs can use the PL/I ADDR function.

```
►►─┬─ DECLARE ─┬─ ADDR BUILTIN;───────────────────────────────────────►◄
   └─ DCL ─────┘
```

4.7 DECLARE ABORT

Include the ABORT ENTRY OPTIONS statement to specify entry options for ABORT.

```
►►──┬─ DECLARE ─┬─ ABORT ENTRY OPTIONS (INTER, ASSEMBLER); ──────────────────►◄
    └─ DCL ─────┘
```

4.8 DECLARE IDMSP

Include the IDMSP ENTRY statement if your online application passes parameters using the TRANSFER statement.

```
►►─┬─ DECLARE ─┬─ IDMSP ENTRY; ─────────────────────────────────────────►◄
   └─ DCL ─────┘
```

Chapter 5. DML Precompiler-Directive Statements

5.1 Overview

This chapter describes the DML precompiler-directive statements. With the precompiler-directive statements, you instruct the DML precompiler to copy source code from the dictionary into your PL/I application program.

If your program accesses the database, it invokes a subschema and issues DML statements. Therefore, it *must* include at least a DECLARE SUBSCHEMA statement. This statement identifies the subschema your program uses and the operating environment in which it executes. If your program includes a DECLARE SUBSCHEMA statement, the DML precompiler automatically generates required source-code components, so you can omit all other precompiler-directive statements.

If your program does not access the database, it does not require DML precompiler-directive statements.

Note: In this chapter, references to the IDMS communications block apply to both the IDMS-DB and IDMS-DC communications blocks.

5.2 DECLARE SUBSCHEMA

Application programs that access the database require the DECLARE SUBSCHEMA statement. This statement:

- Identifies a subschema view to the DML precompiler. The subschema that you name in this statement determines the CA-IDMS/DB record descriptions that the DML precompiler can copy into your program from the data dictionary.
- Identifies your program to the DML precompiler.
- Identifies the operating mode (protocol) and environment under which the program executes. The operating mode determines the form and content of calling sequences produced by the DML precompiler.
- Specifies whether to number each DML command for identification during error reporting (debug sequencing).

Syntax

```
►►─── DECLARE ───────────────────────────────────────────────────────────►

►─┬─────────────────────────────────────────────────────────────────────┬─►─
  └─ (subschema-name SUBSCHEMA, schema-name SCHEMA ─────────────────────

─►─┬─────────────────────────────────────────────────────────────────────┬─►─
   └─ VERSION version-number ─┘

─►─┬─────────────────────────────────────────────────────┬─ ) ─┬─────────►
   └─ , program-name PROGRAM ─┬──────────────────────────┤
                              └─ VERSION version-number ─┘

►─┬─────────────────────────────────┬────────────────────────────────────►
  └─ MODE ( ─┬─ BATCH ← ───┬─ ) ─┘
             ├─ IDMS_DC ───┤
             ├─ DC_BATCH ──┤
             └─ mode ──────┘

►─┬───────────┬──────────────────────────────────────────────────────────►
  └─ DEBUG ─┘

►─┬───────────────────────────────────────────────┬─ ; ──────────────────►◄
  └─ SUBSCHEMA_NAMES LENGTH ( ─┬─ 16 ─┬─ ) ─┘
                               └─ 18 ─┘
```

Parameters

<u>subschema-name</u> SUBSCHEMA,<u>schema-name</u> SCHEMA

Specifies the subschema and schema view of the database used by your program. The subschema and schema definitions must already exist in the data dictionary. If your DBA preregisters program names valid for the subschema in the data dictionary, the program name that you specify in the *program-name* parameter (described below) must be associated with this subschema in the dictionary.

VERSION **<u>version-number</u>**

Optionally qualifies *schema-name* with a version number. *Version-number* must be an integer in the range 1 through 9999. The default is the highest version number defined in the data dictionary for *schema-name*.

<u>program-name</u> **PROGRAM**

Optionally specifies the name of your program. If you preregistered this program in the data dictionary, make sure that *program-name* matches the name in the data dictionary. Otherwise, the DML precompiler will not recognize the program.

VERSION **<u>version-number</u>**

Optionally qualifies *program-name* with a version number (for example, for purposes of testing or development). *version-number* must be an integer in the range 1 through 9999. *Version-number* defaults to the highest number defined in the data dictionary for the program, or defaults to 1 if the program is not registered in the dictionary.

MODE

Identifies the operating mode used by the DML precompiler to generate call statements for the program's DML statements.

BATCH

Specifies that your program executes in batch mode. The DBMS copies the IDMS-DB communications block into program variable storage and generates standard CALL sequences. BATCH is the default.

IDMS_DC

Specifies that your program executes in IDMS_DC mode. The DBMS copies the IDMS-DC communications block into program variable storage and generates CA-IDMS/DC CALL sequences for CA-IDMS/DC requests.

DC_BATCH

Specifies that your program executes in DC-BATCH mode. The DBMS copies the IDMS-DC communications block into program variable storage and generates DC_BATCH CALL sequences for CA-IDMS/DC requests.

DC_BATCH allows you to use all of the database DML commands, and also the following CA-IDMS/DC DML commands:

BIND
COMMIT TASK
DELETE QUEUE
FINISH
GET QUEUE
PUT QUEUE
ROLLBACK
WRITE PRINTER

You specify MODE DC_BATCH to access CA-IDMS/DC queues and printers from batch applications running under the DC/UCF system.

<u>mode</u>

Indicates that your program executes in a special environment, determined by the database administrator. Special environments include user-defined operating modes and teleprocessing monitors. The DML precompiler copies the appropriate communications block into program variable storage and generates operating-mode-specific CALL sequences.

Acceptable values for *mode* are:

CICS
CICS_EXEC
INTERCOMM
PL1F
PL1OPT
SHADOW
TASKMASTER

DEBUG

Instructs the DML precompiler to place a unique DML sequence number in the IDMS communications block for each DML statement. These numbers appear in columns 82 through 89 of the PL/I compiler output listing, in the form DMLP*nnnn*. The DML precompiler generates numbers to identify the sequence in which DML statements appear in the program. Depending on the error routine defined by the DBA, you can use the DML sequence number to help debug your program.

If you do not specify DEBUG, the DML precompiler does not associate sequence numbers with source statements.

16/18

Specifies either 16 bytes or 18 bytes for the following fields in the IDMS communications block: RECORD_NAME, AREA_NAME, ERROR_SET, ERROR_RECORD, and ERROR_AREA.

Example: The following example illustrates how to use the DECLARE SUBSCHEMA statement. In this example, DECLARE SUBSCHEMA accesses the EMPSS09 subschema of the EMPSCH schema for a program named PLITST. The program runs under the IDMS_DC operating mode and includes DEBUG sequencing.

```
DECLARE (EMPSS09 SUBSCHEMA,EMPSCHM SCHEMA,PLITST PROGRAM)
        MODE (IDMS_DC)
                DEBUG;
```

5.3 DECLARE MAP

The DECLARE MAP statement:

- Indicates to the DML precompiler that your program uses mapping-mode terminal I/O
- Defines the program's maps

Repeat the DECLARE MAP statement as many times as required to define each map used by your program. Code DECLARE MAP statements for all of your maps before the first INCLUDE IDMS statement.

Syntax

```
►►── DECLARE (map-name MAP ─┬────────────────────────────┬─ ) ──────────►
                            └─ VERSION version-number ─┘

►─┬───────────────────────────────────────────────┬─ ; ─────────────►◄
  └─ TYPE ( ─┬─ STANDARD ← ─┬─ ) ─┬────────────┬─┘
             └─ EXTENDED ───┘     └─ PAGING ─┘
```

Parameters

map-name MAP
: Specifies the name of a map used by the program. *Map-name* must be the 1- to 8-character name of a map defined in the dictionary.

VERSION version-number
: Optionally qualifies the named map with a version number. *Version-number* must be an integer in the range 1 through 9999 that is associated with the named map in the data dictionary.

TYPE
: Specifies whether the map request block (MRB) built for the map will be standard or extended.

STANDARD
: Specifies that the map has standard 3270 terminal attributes. STANDARD is the default.

EXTENDED
: Specifies that the map has extended 3279 terminal attributes. You can use such mapping features as color, blinking fields, and reverse video for your application programs running under 3279-type terminals.

PAGING
: Specifies that the named map is a pageable map. For more information on pageable maps, see 6.49, "MAP IN (DC/UCF)" on page 6-124 , and 6.50, "MAP OUT (DC/UCF)" on page 6-129, or refer to the *CA-IDMS Mapping Facility*.

Example: The following example illustrates how to use the DECLARE MAP statement to access the EMPMAPLR map:

```
DECLARE (EMPMAPLR MAP);
```

5.4 INCLUDE IDMS

You can code INCLUDE IDMS statements in your application program to copy source code into the program. The data dictionary contains one or more items of source code that correspond to each INCLUDE IDMS statement parameter. Accordingly, your choice of parameters determines the items of code copied from the data dictionary into your program. The syntax rules for INCLUDE IDMS (shown below) describe the INCLUDE IDMS statement parameters with their associated items of source code.

The source code that you copy into your program depends on the usage mode defined in the program's subschema. The subschema usage modes are DML, LR, and MIXED. These usage modes determine your program's source code requirements; thus, they determine whether the program can access database records only, logical records only, or both database records and logical records. Do not code INCLUDE IDMS statements to copy items that conflict with your program's subschema usage mode. For example, do not code SUBSCHEMA_LR_CTRL if your program's subschema usage mode is DML.

Subschema usage modes: The following table describes subschema usage modes and the source code each requires.

Subschema usage mode	Description and required source code
DML	Allows a program to access database records only. DML requires the following source code items: ■ The IDMS communications block through which the application program and the DBMS communicate. For more details, see Chapter 3, "Communications Blocks and Error Detection" on page 3-1. ■ The descriptions of the records to which the subschema permits access.
LR	Allows a program to access logical records only. LR requires the following source code items: ■ The IDMS communications block through which LRF and the DBMS communicate. For more details, see Chapter 3, "Communications Blocks and Error Detection" on page 3-1. ■ The logical-record request control (LRC) block through which the application program and LRF communicate. For more details, see Chapter 3, "Communications Blocks and Error Detection" on page 3-1. ■ The descriptions of the logical records contained in the subschema.

Subschema usage mode	Description and required source code
MIXED	Allows a program to access both database records and logical records. MIXED requires the following source code items: ■ The IDMS communications block, through which LRF and the DBMS communicate. For more details, see Chapter 3, "Communications Blocks and Error Detection" on page 3-1. ■ The description of all records to which the subschema permits access. ■ The logical-record request control (LRC) block, through which the application program and the Logical Record Facility communicate. For more details, see Chapter 3, "Communications Blocks and Error Detection" on page 3-1. ■ The descriptions of all logical records contained in the subschema. Usage of MIXED mode is not recommended for the following reasons: ■ Issuing both logical-record and database requests requires that your program take into account the database currencies maintained in the paths used to service logical-record requests. ■ Accessing both logical records and database records in the same program can diminish the program's independence from the database structure. This could interfere with the execution of paths invoked to provide requested logical-record access. ■ Logical-record path processing can interfere with program access to database records. You may need to insert a DML statement after a logical-record request to reestablish the appropriate currency.

Syntax

```
►►─┬──────────────┬─ INCLUDE IDMS ──────────────────────────────────────►
   └─ level-number ┘

 ►─┬─ (─┬─ SUBSCHEMA_DML_LR_DESCRIPTION ─┬─┬───────────┬─┬─) ─┬─ ; ─►◄
   │    ├─ SUBSCHEMA_DESCRIPTION ────────┤ └─ attribute ┘ │    │
   │    ├─ SUBSCHEMA_CTRL ───────────────┤                │    │
   │    ├─ SUBSCHEMA_RECORDS ────────────┘                │    │
   │    ├─ record-name ─┬──────────────────┬─┬───────────┬┤    │
   │    │               └─ version-number ─┘ └─ attribute ┘│    │
   │    ├─ SUBSCHEMA_LR_DESCRIPTION ─┬─┬───────────┬───────┤    │
   │    ├─ SUBSCHEMA_LR_CONTROL ─────┤ └─ attribute ┘       │    │
   │    ├─ SUBSCHEMA_LR_RECORDS ─────┘                      │    │
   │    ├─ LR (logical-record-name) ─┬───────────┬──────────┤    │
   │    │                            └─ attribute ┘          │    │
   │    ├─ MAPS ─────────────────────┬─┬───────────┬─────────┘    │
   │    ├─ MAP map-name ─────────────┤ └─ attribute ┘              │
   │    ├─ MAP_CONTROLS ─────────────┤                            │
   │    ├─ MAP_CONTROL map-name ─────┤                            │
   │    └─ MAP_RECORDS ──────────────┘                            │
   └─ (SUBSCHEMA_LR_CTRL) ─┬──────────────────────┬─┬───────────┬─┘
                           └─ SIZE lrc-block-size ─┘ └─ attribute ┘
```

Parameters

level-number INCLUDE IDMS

Instructs the DML precompiler to copy source code into your program at the INCLUDE IDMS statement's location.

The optional *level-number* clause instructs the DML precompiler to copy descriptions into your program at a different level than the level specified in the data dictionary. *Level-number* must be an integer in the range 01 through 99. If your program specifies *level-number*, the DML precompiler copies the first level of code to the level specified by *level-number* and adjusts all other levels accordingly. If your program does not specify *level-number*, the descriptions copied by the DML precompiler have the same level numbers as originally specified in the dictionary.

Using the *level-number* clause can cause unpredictable results if record fields are defined with a SYNCHRONIZED clause. Such fields may contain slack bytes, inserted to ensure correct alignment. Because CA-IDMS/DB and CA-IDMS/DC don't regard slack bytes as functional, fields that contain such bytes may be misrepresented. Therefore, you should ensure that all fields and records are structured properly.

SUBSCHEMA_DML_LR_DESCRIPTION

Copies all components required to access both database and logical records:

SUBSCHEMA_CTRL

SUBSCHEMA_RECORDS

SUBSCHEMA_LR_CTRL

SUBSCHEMA_LR_RECORDS

You specify SUBSCHEMA_DML_LR_DESCRIPTION only if the subschema usage mode is MIXED. Do not specify SUBSCHEMA_DML_LR_DESCRIPTION if the usage mode is DML or LR.

SUBSCHEMA_DESCRIPTION
Copies all components required to access database records:

SUBSCHEMA_CTRL

SUBSCHEMA_RECORDS

Do not specify SUBSCHEMA_DESCRIPTION if the subschema usage mode is LR.

SUBSCHEMA_CTRL
Copies the IDMS-DB communications block data description. If the operating mode is IDMS_DC or DC_BATCH, SUBSCHEMA_CTRL copies the IDMS-DC communications block.

SUBSCHEMA_RECORDS
Copies the descriptions of all records contained in the subschema. The DML precompiler may copy into your program PL/I synonyms defined for the subschema records in the data dictionary, according to the rules of synonym usage. Do not specify SUBSCHEMA_RECORDS if the subschema usage mode is LR.

Note: When copying a schema-owned record, the DML precompiler adds up to 7 bytes, if necessary, to make the record length divisible by 8 for doubleword alignment.

<u>record-name</u> VERSION <u>version-number</u> <u>attribute</u>
Copies the description of a record defined in the dictionary. Do not specify record if the subschema's usage mode is LR.

<u>record-name</u>
Specifies the name of the record to be copied. It can be the primary name of a record stored in the data dictionary, or a synonym.

Schema-owned records cannot be copied into non CA-IDMS programs. These are programs that neither use a subschema nor access the database. However, a synonym defined for a schema-owned record *can* be copied into a non CA-IDMS program. You use the VERSION clause to identify the synonym.

If the DMLP processor cannot find a record named *record-name* in the dictionary, it searches for a module by that name. The module, which may have been stored using the DDDL compiler, presumably contains a definition of records not included in the subschema. If an operating mode is associated with the named record or module in the data dictionary, it must agree with the mode in effect for your program. (See "DECLARE SUBSCHEMA", earlier in this chapter.) For more information on associating operating modes with records, refer to the *IDD DDDL Reference*.

VERSION <u>version-number</u>

Optionally qualifies IDD records, but not schema-owned records, with a version number. *Version-number* must be an integer in the range 1 through 9999. *Version-number* defaults to the highest version number of the record defined in the data dictionary for the language and operating mode under which the program compiles.

<u>attribute</u>

Optionally allows you to instruct the DML precompiler to include PL/I attributes in the PL/I DECLARE statement. The DML precompiler generates the PL/I DECLARE statement for the record that you specify in *record-name*.

SUBSCHEMA_LR_DESCRIPTION

Copies all components required to access logical records:

SUBSCHEMA_CTRL

SUBSCHEMA_LR_CTRL

SUBSCHEMA_LR_RECORDS

Do not specify SUBSCHEMA_LR_DESCRIPTION if the subschema's usage mode is DML.

SUBSCHEMA_LR_CONTROL

Copies the SUBSCHEMA_CTRL and SUBSCHEMA_LR_CTRL components. Do not specify SUBSCHEMA_LR_CONTROL if the subschema usage mode is DML.

SUBSCHEMA_LR_RECORDS

Copies the descriptions of all logical records defined in the subschema. All participating database records become 02-level group fields. This allows your program to reference the portion of a logical record corresponding to a database record as a group field. Do not specify SUBSCHEMA_LR_RECORDS if the subschema usage mode is DML.

Note: When copying a schema-owned record, the DML precompiler adds up to 7 bytes, if necessary, to make the record length divisible by 8 for doubleword alignment.

LR (<u>logical-record-name</u>)

Copies the description of an individual logical record contained in the subschema: do not include LR if the subschema usage mode is DML.

<u>logical-record-name</u>

Names the logical record.

<u>attribute</u>

Optionally allows you to instruct the DML precompiler to include PL/I attributes in the PL/I DECLARE statement. The DML precompiler generates the PL/I DECLARE statement for the logical record that you specify in *logical-record-name*.

MAPS

Copies the map request block (MRB) and map records for the maps that you specify with DECLARE MAP statements.

MAP

Copies the MRB and map records associated with the named map. The map's version number defaults to the version number that you specify for this map in the DECLARE MAP statement.

map-name

Names the map.

attribute

Attribute optionally allows you to instruct the DML precompiler to include PL/I attributes in the PL/I DECLARE statement. The DML precompiler generates the PL/I DECLARE statement for the map that you specify in *map-name*.

MAP_CONTROLS

Copies the MRBs for the maps that you specify in DECLARE MAP statements.

MAP_CONTROL

Copies the MRB for the named map. The map's version number defaults to the version number that you specify for this map in the DECLARE MAP statement.

map-name

Names a map.

attribute

Optionally allows you to instruct the DML precompiler to include PL/I attributes in the PL/I DECLARE statement. The DML precompiler generates the PL/I DECLARE statement for the map that you specify in *map-name*.

MAP_RECORDS

Copies the map records for the maps that you specify in DECLARE MAP statements.

SUBSCHEMA_LR_CTRL

Copies the LRC block data description.

Do not specify SUBSCHEMA_LR_CTRL if the subschema usage mode is DML.

SIZE (lrc-block-size)

Optionally specifies the size of that portion of the LRC block that contains information about the logical-record-request WHERE clause (PXE).

Lrc-block-size defaults to 512 bytes. If you include *lrc-block-size*, you should specify a size large enough to accommodate the most complex WHERE clause in the program. The default, 512, is large enough to include approximately 32 operators, operands, and literals.

Lrc-block-size must be a positive integer in the range 0 through 9999. You specify a value of 0 if none of the logical-record requests issued by the program includes a WHERE clause. You calculate *lrc-block-size* as follows:

1. Multiply the greatest number of operands and operators in a single WHERE clause by 16 bytes.

2. Add the number of bytes, rounded up to the nearest multiple of 8, associated with the data field for each operand that is a keyword, a program variable, or a logical-record field named in the OF LR clause.

3. Add the length, rounded up to the nearest multiple of 8, of each operand that is a character literal.

4. Add 12 bytes for each operand that is a numeric literal.

INCLUDE IDMS code: The following figure shows the code that the DML precompiler copies into program variable storage for each INCLUDE IDMS statement parameter.

source code components brought in from the data dictionary by the DMLP Processor

INCLUDE IDMS Statements: INCLUDE IDMS	SUBSCHEMA_CTRL	SUBSCHEMA_RECORDS	*record-name*		SUBSCHEMA_LR_CTRL	SUBSCHEMA_LR_NAMES	SUBSCHEMA_LR_RECORDS	*logical-record-name*
SUBSCHEMA_DML_LR_DESCRIPTION	X	X			X		X	
SUBSCHEMA_DESCRIPTION	X	X						
SUBSCHEMA_CTRL	X							
SUBSCHEMA_RECORDS		X						
record-name			X					
SUBSCHEMA_LR_DESCRIPTION	X				X	X	X	
SUBSCHEMA_LR_CONTROL	X				X	X		
SUBSCHEMA_LR_CTRL					X			
SUBSCHEMA_LR_RECORDS							X	
LR *logical-record-name*								X

5.5 INCLUDE IDMS (MAP_BINDS)

INCLUDE IDMS (MAP_BINDS) copies map- and map-record-specific BIND MAP statements for all maps that you specify with DECLARE MAP statements.

Syntax

```
►►─┬──────────────────────────────┬─────────────────────────────────────►◄
   └─ INCLUDE IDMS (MAP_BINDS); ─┘
```

Parameters

INCLUDE IDMS (MAP_BINDS)
Copies map- and map-record-specific BIND MAP statements for all maps that you specify with DECLARE MAP statements.

If your program uses a map, it requires a BIND MAP statement for the map and for each associated map record. The BIND MAP statement identifies the location of the MRB and initializes fields within the MRB. If you code the INCLUDE IDMS (MAP_BINDS) statement in your program, the DML processor automatically copies appropriate BIND MAP statements into your program. For more information on the BIND MAP statement, see 6.14, "BIND MAP (DC/UCF)" on page 6-30.

You must individually bind map records associated with logical records.

5.6 INCLUDE IDMS MODULE

INCLUDE IDMS *(module-name)* copies procedure source statements defined by the database administrator as modules in the dictionary.

Syntax

```
      ┌──────────────────────────────────────────────────────────────┐
►►─┬──↓──────────────────────────────────────────────────────────────┴──────►◄
   └─ INCLUDE IDMS (module-name ─┬──────────────────────────┬─ ); ─┘
                                 └─ VERSION version-number ─┘
```

Parameters

INCLUDE IDMS (<u>module-name</u>)

Copies procedure source statements defined by the DBA as modules in the dictionary. *Module-name* specifies the name of a module previously defined by the DBA using the DDDL compiler (refer to the *IDD DDDL Reference*). The available PL/I standard modules are:

- IDMS_STATUS
- IDMS_STATUS (mode is IDMS_DC)

The DML precompiler inserts the module into your program at the location of the INCLUDE IDMS MODULE statement, without modification. If the module contains DML statements, the DML precompiler examines and expands them within the context of your program's subschema view and compile mode, as if they were coded directly.

Note: The INCLUDE IDMS MODULE statement can precede the DECLARE SUBSCHEMA statement if the module it copies does not contain DML statements.

You can nest INCLUDE IDMS MODULE statements. This means that code invoked by an INCLUDE IDMS MODULE entry can itself contain INCLUDE IDMS MODULE statements. However, make sure that a copied module does not copy itself.

VERSION <u>version-number</u>

Optionally qualifies *module-name* with a version number. *Version-number* must be an integer in the range 1 through 9999.

There are two defaults for *version-number*, depending on whether:

- There is a version of the module that you name with *module-name* which is operating-mode-specific. In this case, the default is the version number of this module. If there are two or more mode-specific versions of the module, *version-number* defaults to the highest version number among these versions.
- There is a version of the module that you name with *module-name* which is non-operating-mode-specific, and there exists no operating-mode-specific version. In this case, the default is the version number of this module. If there are two or more non-mode-specific versions of the module, *version-number* defaults to the highest version number among these versions.

If no version of the module exists in the dictionary, an error condition results. For more information, refer to *CA-IDMS Messages and Codes*.

5.7 INCLUDE IDMS (SUBSCHEMA_BINDS)

INCLUDE IDMS (SUBSCHEMA_BINDS):

- Initializes the PROGRAM_NAME field in the IDMS-DB communications block
- Copies a standard BIND RUN_UNIT statement and appropriate standard BIND RECORD commands for each CA-IDMS/DB record in your program's variable storage.

This statement does not generate BIND RECORD statements for logical records. Your program does not need them. INCLUDE IDMS (SUBSCHEMA_BINDS) only generates BINDS for subschema records explicitly copied into your program by INCLUDE IDMS statements.

Do not use the INCLUDE IDMS (SUBSCHEMA_BINDS) statement when binding several records to the same location. Instead, code BIND RUN_UNIT and BIND RECORD statements separately for each record. This allows you to include a CALL IDMS_STATUS statement after each BIND RECORD statement to check the ERROR_STATUS field.

Note: The INCLUDE IDMS (SUBSCHEMA_BINDS) statement does not automatically generate BIND RECORD statements when more than one copy of a given database record description (including synonyms) is present in the program. For such records, issue individual BIND RECORD statements to bind the records to the correct location.

Syntax

```
►►─┬──────────────────────────────────┬──────────────────────────►◄
   └─ INCLUDE IDMS (SUBSCHEMA_BINDS); ─┘
```

5.8 INCLUDE IDMS (SUBSCHEMA_RECORD_BINDS)

INCLUDE IDMS SUBSCHEMA_RECORD_BINDS copies appropriate standard BIND *record-name* statements for each CA-IDMS/DB record in the program.

In cases where more than one copy of a given database record description (including synonyms) is present in the program, INCLUDE IDMS SUBSCHEMA_RECORD_BINDS will not automatically generate bind record statements. Individual bind record statements must be issued to bind the record to the correct location.

Do not use the INCLUDE IDMS SUBSCHEMA_RECORD_BINDS statement when binding several records to the same location. Instead, code DML BIND statements for each record.

Syntax

```
►►─┬──────────────────────────────────────────┬──────────────────►◄
   └─ INCLUDE IDMS (SUBSCHEMA_RECORD_BINDS); ─┘
```

Chapter 6. Data Manipulation Language Statements

6.1 Overview

This chapter describes the Data Manipulation Language (DML) that applies to CA-IDMS/DB, CA-IDMS/DC, and CA-IDMS/UCF.

Note: The DC/UCF references in this chapter include both the CA-IDMS/DC and CA-IDMS/UCF products.

DML consists of statements that enable you to access the database management system (DBMS) and to request Logical Record Facility (LRF) and data communications services.

This chapter presents the following information:

- Tables describing the database and data communications functions of DML statements
- Tables grouping the DML statements by function
- Discussions of each DML statement (statements are in alphabetical order). Discussions include an overall description of the statement, syntax, parameter descriptions, and examples

  Important: When you review the syntax for each DML statement, note that you must code the parameters in the order in which they are shown.

6.2 Functions of DML statements

This section describes the 14 categories of DML statements. There are 6 categories of database (CA-IDMS/DB) functions. There are 8 categories of data communications (DC/UCF system) functions.

Database functions: The following is a list of the 6 database DML functions:

- **Control** statements:
  - Initiate and terminate processing
  - Effect recovery
  - Prevent concurrent retrieval and update of database records
  - Evaluate set conditions
- **Retrieval** statements locate records in the database and make them available to the application program.
- **Modification** statements add new records to the database and modify and delete existing records.
- **Accept** statements move special information such as database keys, storage addresses, and statistics from the DBMS to program variable storage.
- **Logical-record** statements retrieve, modify, store, and erase logical records.
- **Recovery** statements perform functions relating to database, scratch, and queue area recovery in the event of a system failure. These functions:
  - Establish checkpoints in the journal file for database, scratch, and queue records used by the issuing task
  - Roll back user database, scratch, and queue areas to the last checkpoint established
  - Establish an end-of-task checkpoint and relinquish control of all database, scratch, and queue areas associated with the issuing task
  - Write user-defined records to the journal file

Data communications functions: The following is a list of the 8 data communications DML functions:

- **Program management** statements:
  - Pass and return control from one program to another
  - Load and delete programs and tables
  - Define exit routines to be performed before an abnormal program termination (abend)
  - Force an abend condition
- **Storage management** statements allocate and release variable storage.

- **Task management** statements:
  - Initiate a new task
  - Change the dispatching priority of the issuing task
  - Enqueue and dequeue system resources
  - Signal that a task is to wait pending completion of an event
  - Post an event control block (ECB), indicating completion of an event
- **Time management** statements obtain the time and date and define time-related events. These events include:
  - Placing the issuing task in a wait state for a specified amount of time
  - Posting a user-specified ECB after a specified interval
  - Initiating a new task after a specified interval
- **Scratch management** statements create, delete, or retrieve records from the scratch area.
- **Queue management** statements create, delete, or retrieve records from the queue area.
- **Terminal management** statements transfer data between the application program and the terminal.
- **Utility function** statements:
  - Request retrieval of task-related information
  - Request a memory dump of selected parts of storage
  - Retrieve and send a predefined message stored in the data dictionary
  - Send a specified message to one or more users or logical terminals
  - Collect, retrieve, and write DC/UCF system statistics on a transaction basis
  - Establish longterm database locks and monitor access to database records used across tasks during a pseudo-conversational transaction

6.3 DML statements grouped by function

The two tables in this section list and describe the DML statements by their database and data communications functions, respectively.

6.3.1 DML statements (database)

The following table lists CA-IDMS/DB DML statements by function.

Note: You can use CA-IDMS/DB statements in a DC/UCF system environment. However, you cannot use DC/UCF system statements in the CA-IDMS/DB environment.

Function	DML Statement	Description
Control	BIND RUN-UNIT	Signs on the application program to the DBMS
	BIND RECORD	Establishes addressability in variable storage for one or more records included in the program's subschema
	BIND PROCEDURE	Establishes communication between the application program and a DBA-defined database procedure
	READY	Prepares database areas for processing
	FINISH	Releases database areas from program control
	IF	Evaluates the presence of member records in a set or a record's membership status and specifies action based on the outcome
	COMMIT	Writes a checkpoint to the journal file and releases record locks
	ROLLBACK	Requests recovery of database, scratch, and queue areas
	KEEP	Places locks on record occurrences

Function	DML Statement	Description
Retrieval	FIND/OBTAIN DBKEY	Accesses a record using a db-key previously saved by the program
	FIND/OBTAIN CURRENT	Accesses a record using previously established currencies
	FIND/OBTAIN WITHIN SET/AREA	Accesses a record based on its logical location within a set or its physical location within an area
	FIND/OBTAIN OWNER	Accesses the owner record of a set occurrence
	FIND/OBTAIN CALC/DUPLICATE	Accesses a record using its CALC-key value
	FIND/OBTAIN USING SORT KEY	Accesses a record in a sorted set using its sort-key value
	GET	Moves all data associated with a previously located record into program variable storage
	RETURN	Retrieves the database and symbolic keys of an indexed record entry
Modification	STORE	Adds a new record to the database
	MODIFY	Changes the contents of an existing record
	CONNECT	Links a record to a set
	DISCONNECT	Removes a member record from a set
	ERASE	Deletes a record from the database
Accept	ACCEPT DBKEY FROM CURRENCY	Saves the db-key of the current record of run unit, record type, set, or area
	ACCEPT DBKEY RELATIVE TO CURRENCY	Saves the db-key of the next, prior, or owner record relative to the current record of a set

Function	DML Statement	Description
	ACCEPT IDMS STATISTICS	Returns system runtime statistics to the program
	ACCEPT BIND RECORD	Returns a record's bind address to the program
	ACCEPT PAGE_INFO	Returns page information for a given record to the program
	ACCEPT PROCEDURE	Returns information from the application program information block associated with a database procedure to the program
Logical Record Facility	ERASE	Deletes a logical record
	MODIFY	Modifies a logical record
	OBTAIN	Accesses a logical record
	STORE	Stores a logical record
Recovery	COMMIT	Establishes a checkpoint in the journal file for database, scratch, and queue record activity
	FINISH	Relinquishes control of database, scratch, and queue areas
	ROLLBACK	Rolls back database, scratch, and queue areas to the last checkpoint
	WRITE JOURNAL	Writes user-defined records to the journal file

6.3.2 DML statements (data communications)

The following table lists DC/UCF DML statements by function.

Note: You cannot use DC/UCF system statements in the CA-IDMS/DB environment.

Function	DML Statement	Description
Program Management	TRANSFER (LINK)	Passes control to another program with the expectation of receiving it back

Function	DML Statement	Description
	TRANSFER (XCTL)	Passes control to another program with no expectation of receiving it back
	DC RETURN	Returns control to the next higher level calling program
	LOAD TABLE	Loads a program or table into the DC/UCF program pool
	DELETE TABLE	Signals that a program has finished using a program or a table in the program pool
	ABEND	Abnormally terminates the issuing task
Storage Management	GET STORAGE	Allocates variable storage from an DC/UCF storage pool
	FREE STORAGE	Frees all or part of a block of variable storage
Task Management	ATTACH	Attaches a new task within DC/UCF
	CHANGE PRIORITY	Changes the dispatching priority of the issuing task
	ENQUEUE	Acquires a resource or a list of resources
	DEQUEUE	Releases a resource
	WAIT	Relinquishes control to DC/UCF while awaiting completion of an event
	POST	Posts an event control block (ECB)
Time Management	GET TIME	Obtains the time and date from the system
	SET TIMER	Defines a time-delayed event
Scratch Management	PUT SCRATCH	Stores a scratch record
	GET SCRATCH	Retrieves a scratch record
	DELETE SCRATCH	Deletes a scratch record

Function	DML Statement	Description
Queue Management	PUT QUEUE	Stores a queue record
	GET QUEUE	Retrieves a queue record
	DELETE QUEUE	Deletes a queue record
Terminal Management (Basic Mode)	READ TERMINAL	Requests a synchronous or asynchronous data transfer from the terminal to program variable storage
	WRITE TERMINAL	Requests a synchronous or asynchronous data transfer from program variable storage to the terminal buffer
	WRITE THEN READ TERMINAL	Requests a synchronous or asynchronous data transfer from program variable storage to the terminal buffer; and on a terminal operator signal, back to variable storage
	CHECK TERMINAL	Ensures that a previously issued asynchronous I/O operation is complete
Terminal Management (Line Mode)	READ LINE FROM TERMINAL	Requests a synchronous data transfer from the terminal to the issuing program
	WRITE LINE TO TERMINAL	Requests a synchronous or asynchronous data transfer from the issuing program to the terminal
	END LINE TERMINAL SESSION	Terminates the current line I/O session
	WRITE PRINTER	Requests transmission of data from a task to a printer
Terminal Management (Mapping Mode)	MAP IN	Requests a transfer of data from the terminal to program variable storage
	MAP OUT	Requests a transfer of data from program variable storage to the terminal

Function	DML Statement	Description
	MAP OUTIN	Requests a transfer of data from program variable storage to the terminal; and, upon a terminal operator signal, back to variable storage
	INQUIRE MAP	Obtains information or tests conditions concerning the previous mapping operation
	MODIFY MAP	Requests modifications of mapping options for a map
	STARTPAGE	Begins a map paging session and specifies options for that session
	ENDPAGE	Terminates a map paging session
Utility	BIND MAP	Identifies the location of a map request block (MRB) and initializes the MRB's fields
	ACCEPT	Retrieves task-related information
	SNAP	Requests a memory dump of selected parts of storage
	SEND MESSAGE	Sends a message to a user, logical terminal, or list of users or logical terminals
	BIND TRANSACTION STATISTICS	Defines the beginning of a transaction for the purpose of collecting transaction statistics
	ACCEPT TRANSACTION STATISTICS	Returns the contents of the transaction statistics block (TSB) to program variable storage
	END TRANSACTION STATISTICS	Defines the end of a transaction

Function	DML Statement	Description
	KEEP LONGTERM	Either modifies a prior KEEP LONGTERM request or enables database longterm locks or database monitoring for records, sets, or areas
	WRITE LOG	Retrieves a message from the data dictionary and sends it to a predefined destination

6.4 ABEND (DC/UCF)

The ABEND statement terminates the issuing task abnormally. Optionally, ABEND also writes a task dump to the log file. Upon completion of the ABEND function, the DBMS returns processing control to the DC/UCF system program-control module.

Syntax

```
►►── ABEND CODE (abend-code) ─┬────────────┬─ ; ─────────────────────────►◄
                              ├─ NODUMP ← ─┤
                              └─ DUMP ─────┘
```

Parameters

ABEND CODE(abend-code)
Specifies a 4-character abend code that you select. *Abend-code* can be the symbolic name of a variable storage field containing the abend code, or the code itself enclosed in single quotation marks.

Note: Because the abend code that you specify appears in the system log and displays at the task's terminal, you should not use system abend codes.

NODUMP/DUMP
Specifies whether the system writes a formatted task dump to the log file. The default is NODUMP.

Example: In this example, ABEND terminates the issuing task abnormally, issuing the code U876, and writes a task dump to the log file:

```
ABEND CODE('U876')
   DUMP;
```

Status codes: Because the DBMS passes control to the system program-control module, your program does not have to check the ERROR_STATUS field.

6.5 ACCEPT (DC/UCF)

The ACCEPT statement retrieves the following task-related information:

- Current task code
- Task identifier
- Logical terminal identifier
- Physical terminal identifier
- DC/UCF system version
- User identifier (the ID of the user signed on to the task's logical terminal)
- Physical terminal screen dimensions

Syntax

```
►►─── ACCEPT ──┬─ TASK CODE──┬─ INTO (return-location); ──────────────►◄
               ├─ TASK ID ───┤
               ├─ LTERM ID ──┤
               ├─ PTERM ID ──┤
               ├─ SYSVERSION ┤
               ├─ USER ID ───┤
               └─ SCREENSIZE ┘
```

Parameters.

TASK CODE
Specifies the 1- to 8-character code that invokes the current task.

TASK ID
Specifies the task identifier assigned by the system. The task identifier is a unique sequence number stored in a FIXED BINARY(31) field. At system startup, the DC/UCF system sets the ID to 0. Each time a task executes, the system increments the ID by 1.

LTERM ID
Specifies the 1- to 8-character identifier of the logical terminal associated with the current task. If the current task has no associated logical terminal, the system returns spaces (null value).

PTERM ID
Specifies the 1- to 8-character identifier of the physical terminal associated with the current task. If the current task has no associated physical terminal, the system returns spaces (null value).

SYSVERSION
Specifies the version number of the current DC/UCF system. The version number is an integer in the range 0 through 9999 stored in a halfword binary numeric field.

USER ID
Specifies the 32-character identifier of the user signed on to the logical terminal associated with the current task. If no user is signed on, the system returns spaces (null value).

SCREENSIZE
Specifies the screen dimensions of the current task's associated physical terminal. The system returns the screen size to a field divided into two FIXED BINARY(15) fields. The first field contains the row; the second field contains the column. For example, values of 24 in the first halfword and 80 in the second halfword represent a 24-line by 80-character screen. If the current task has no associated terminal, the system returns a null value of 0.

INTO (return-location)
Specifies the location to which the DC/UCF system returns the requested task-related information. *Return-location* specifies the symbolic name of a user-defined field. The pictures and usages of this field and of the requested data must be compatible.

Example: The following ACCEPT statements illustrate retrieving the ID of the current task and the id of the user signed on to the task's associated logical terminal:

```
ACCEPT TASK ID INTO (TASK_ID);
ACCEPT USER ID INTO (USER_ID);
```

Status codes: Upon completion of the ACCEPT function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
4829	An invalid parameter was passed from the program.

6.6 ACCEPT BIND RECORD

The ACCEPT BIND RECORD statement moves the bind address of a record to a specified location in program variable storage. Usually, a subprogram uses this statement to acquire the address of a record.

Currency: The ACCEPT BIND RECORD statement updates no currencies. However, your program must establish currency for the record type whose bind address it requires.

Syntax:

```
►►── ACCEPT BIND RECORD (record-name) INTO (bind-address); ──────────────►◄
```

record-name
Specifies the record whose bind address will be copied into the specified location in variable storage. *Record-name* must be a record previously bound by the run unit.

INTO (bind-address)
Specifies the variable-storage location to which CA-IDMS/DB and the system return the record's bind address. *Bind-address* is defined as a FIXED BINARY(31) field. After the ACCEPT BIND RECORD statement executes, *bind-address* contains a storage address, not a database key.

Example: This example uses ACCEPT BIND RECORD to move the bind address for the EMPLOYEE record to location REG1 in the requesting subprogram:

```
ACCEPT BIND RECORD (EMPLOYEE) INTO (REG1);
```

Status codes: Upon completion of the ACCEPT BIND RECORD function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
1508	The subschema does not contain the named record.

6.7 ACCEPT DBKEY FROM CURRENCY

The ACCEPT DBKEY FROM CURRENCY statement moves the db-key of the current record of run unit, record type, set, or area to a specified location in program variable storage. By using a FIND/OBTAIN DBKEY statement, you can directly access records whose db-keys you save using the ACCEPT DBKEY FROM CURRENCY statement.

Currency: ACCEPT DBKEY FROM CURRENCY does not update currencies.

Syntax

```
►►── ACCEPT CURRENCY ─┬───────────────────────────┬─ INTO (db-key-field); ───────►◄
                      ├─ RECORD (record-name) ─┤
                      ├─ SET (set name) ───────┤
                      └─ AREA (area-name) ─────┘
```

Parameters

RECORD (record-name)
Saves the db-key of the record current of the specified record type into the location specified by *db-key-field*.

SET (set-name)
Saves the db-key of the record current of the specified set into the location specified by *db-key-field*.

AREA (area-name)
Saves the db-key of the record current of the specified area into the location specified by *db-key-field*.

INTO (db-key-field)
Identifies the location in variable storage that will contain the db-key of the specified record. *Db-key-field* must be a FIXED BINARY(31) field.

Note: If you omit the RECORD, SET, or AREA qualifiers, the DBMS saves the db-key of the record current of run unit.

Example: The following example:

1. Establishes a record, named EMPLOYEE, as current of run unit
2. Saves the record's db-key in a location named SAVED_DBKEY, using the ACCEPT DBKEY FROM CURRENCY statement
3. Accesses the EMPLOYEE record occurrence using the saved db-key

```
EMP_ID_0415 = EMP_ID_IN;
FIND CALC RECORD (EMPLOYEE);
ACCEPT CURRENCY INTO (SAVED_DBKEY);
 .
 .
 .
OBTAIN DBKEY (SAVED_DBKEY);
```

Status codes: Upon completion of the ACCEPT DBKEY FROM CURRENCY function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
1506	Currency was not established for the named record or set.
1508	The subschema does not contain the named record or set. Your program probably invoked the wrong subschema.
1523	The subschema does not contain the named area.

6.8 ACCEPT DBKEY RELATIVE TO CURRENCY

The ACCEPT DBKEY RELATIVE TO CURRENCY statement moves a selected db-key to a specified location in program variable storage. The db-key moved to variable storage can be the db-key of the next, prior, or owner record relative to the current record of set.

This version of the ACCEPT statement allows you to save the db-key of a record within a set without actually having to access the record. By using a FIND/OBTAIN DBKEY statement, you can directly access records whose db-keys you save using the ACCEPT DBKEY RELATIVE TO CURRENCY statement.

Note: You must establish set currency before using this statement. If no set currency is established, the DBMS returns 0000 to the ERROR_STATUS field and -1 to the db-key field.

Currency: ACCEPT DBKEY RELATIVE TO CURRENCY does not update any currencies.

Syntax

```
►►── ACCEPT CURRENCY SET (set name) ─┬─ NEXT ──┬── INTO (db-key-field); ──────►◄
                                     ├─ PRIOR ─┤
                                     └─ OWNER ─┘
```

Parameters

SET (set-name)

Identifies the record whose db-key will be moved into the location specified by *db-key*, described below. *Set-name* must be a set included in the subschema.

When a record declared as an optional or manual member of a set is accessed, it does *not* become current of set unless it is connected to an occurrence of the set. If the record is not connected to an occurrence of the set, an attempt to access the owner record will locate instead the owner of the current record of set. In such cases, use the OWNER option to determine whether the retrieved record is actually a set member before executing the ACCEPT DBKEY RELATIVE TO CURRENCY statement. You can do this with the IF statement, described later in this chapter.

NEXT

Saves the db-key of the next record relative to the record current of the specified set. You cannot request NEXT currency unless the specified set has prior pointers. Prior pointers ensure that the next pointer in the prefix of the current record does not point to a logically deleted record.

No indication of an end-of-set condition is possible for the NEXT or PRIOR options. A retrieval command must be issued to determine whether the next or prior record in the set occurrence is the owner record.

Native VSAM users: You cannot request NEXT currency for sets defined for native VSAM records.

PRIOR
Saves the db-key of the prior record relative to the record current of the specified set. You cannot request PRIOR currency unless the specified set has prior pointers.

No indication of an end-of-set condition is possible for the NEXT or PRIOR options. A retrieval command must be issued to determine whether the next or prior record in the set occurrence is the owner record.

Native VSAM users: You cannot request PRIOR currency for sets defined for native VSAM records.

OWNER
Saves the db-key of the owner of the record current of the specified set. A request for OWNER CURRENCY cannot be executed unless the specified set has owner pointers. However, if the current record of the named set is the owner record occurrence, a request for OWNER currency returns the db-key of the record itself. This will happen even if the set does not have owner pointers.

Native VSAM users: You cannot request OWNER currency for sets defined for native VSAM records.

INTO (db-key-field)
Identifies the location in variable storage that will contain the db-key of the requested record. *Db-key* must be a FIXED BINARY(31) field.

Example: The following statements access the EMP_EXPERTISE set and save the db-key of the owner record of the SKILL_EXPERTISE set:

```
EMP_ID_0415 = '0119';
FIND CALC RECORD (EMPLOYEE);
FIND FIRST SET (EMP_EXPERTISE);
ACCEPT CURRENCY SET (SKILL_EXPERTISE) OWNER
     INTO (SAVE_DBKEY);
```

Status codes: Upon completion of the ACCEPT DBKEY RELATIVE TO CURRENCY function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
1508	The subschema does not contain the named set. Your program probably invoked the wrong subschema.

6.9 ACCEPT IDMS STATISTICS

The ACCEPT IDMS STATISTICS statement copies system runtime statistics located in the program's statistics block to program variable storage. While a run unit executes, your program can issue ACCEPT IDMS STATISTICS as many times as required. For example, you might want to request database statistics after storing a variable-length record. This allows you to determine whether the entire record was stored in one place, or fragments were placed in an overflow area.

The ACCEPT IDMS STATISTICS statement does not reset any of the statistics fields to zero. IDMS statistics block fields are reset only when you issue a FINISH command.

You can use the ACCEPT IDMS STATISTICS statement in both the navigational and Logical Record Facility (LRF) environments.

Syntax

```
►►── ACCEPT IDMS_STATISTICS INTO (db-statistics-field);──────────────────►◄
```

Parameter

db-statistics-field

Identifies the field (in program variable storage) the system runtime statistics contained in IDMS_STATISTICS are to be copied to. *Db-statistics-field* is defined as an aligned, 100-byte field.

The DBMS copies IDMS_STATISTICS data to *db-statistics-field* according to the following format:

```
DECLARE
  01  DB_STATISTICS,
    03  DATE_TODAY              CHAR(8),
    03  TIME_TODAY              CHAR(8),
    03  PAGES_READ              FIXED BINARY(31),
    03  PAGES_WRITTEN           FIXED BINARY(31),
    03  PAGES_REQUESTED         FIXED BINARY(31),
    03  CALC_TARGET             FIXED BINARY(31),
    03  CALC_OVERFLOW           FIXED BINARY(31),
    03  VIA_TARGET              FIXED BINARY(31),
    03  VIA_OVERFLOW            FIXED BINARY(31),
    03  LINES_REQUESTED         FIXED BINARY(31),
    03  RECS_CURRENT            FIXED BINARY(31),
    03  CALLS_TO_IDMS           FIXED BINARY(31),
    03  FRAGMENTS_STORED        FIXED BINARY(31),
    03  RECS_RELOCATED          FIXED BINARY(31),
   *03  LOCKS_REQUESTED         FIXED BINARY(31),
   *03  SEL_LOCKS_HELD          FIXED BINARY(31),
   *03  UPD_LOCKS_HELD          FIXED BINARY(31),
   *03  RUN_UNIT_ID             FIXED BINARY(31),
   *03  TASK_ID                 FIXED BINARY(31),
   *03  LOCAL_ID                CHAR(8),
    03  FILLER                  CHAR(8);

   *Applies to CA-IDMS/DB central version only
```

The LOCAL_ID field consists of the 4-byte identifier of the interface in which the run unit originated (for example, BATC, DBDC, or CICS) and a unique identifier (fullword binary value) assigned to the run unit by that interface. For batch and VM/ESA run units, this identifier specifies the internal machine time. For CICS run units, this identifier specifies the CICS transaction number assigned to the run unit.

To display the originating interface identifier and the run-unit identifier for a program, you can move the LOCAL-ID field to a work field:

```
01   WORK_LOCAL_ID,
  02  WORK_LOCAL_ORIGIN      CHAR(4),
  02  WORK_LOCAL_NUMBER      FIXED BINARY(31);
```

Alternatively, your DBA can modify the DB_STATISTICS record from the data dictionary to define two subordinate fields for the LOCAL_ID field. The DB_STATISTICS record describes the IDMS statistics block. To use this record, code the following statement in program variable storage:

```
01 INCLUDE IDMS (DB_STATISTICS);
```

Example: The following statements:

1. Establish currency for the sets in which a new EXPERTISE record will participate as a member
2. Store the EXPERTISE record
3. Move statistics about the stored EXPERTISE record to the DB_STATISTICS location in main storage

```
EMP_ID_0415 = EMP_ID_IN;
FIND CALC RECORD (EMPLOYEE);
SKILL_ID_IN = SKILL_ID_0455;
FIND CALC RECORD (SKILL);
STORE RECORD (EXPERTISE);
ACCEPT IDMS_STATISTICS INTO (DB_STATISTICS);
```

Status codes: Upon completion of the ACCEPT IDMS STATISTICS function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
1518	The database statistics location was not a valid address.

6.10 ACCEPT PAGE_INFO

The ACCEPT PAGE_INFO statement moves the page information for a given record to a specified location in program variable storage. Page information that is saved in this manner is available for subsequent direct access by using a FIND/OBTAIN DBKEY statement.

Syntax

```
►►─ ACCEPT PAGE_INFO RECORD (record-name) INTO (page-info-location) ─────────►◄
```

Parameters

RECORD (record-name)
Specifies the record whose page information will be placed in the specified location.

INTO (page-info-location)
Specifies the name of the four-byte field that may be defined either as a group field or as a fullword field (PIC S9(8) COMP). Identifies the location in variable storage that contains page information for the specified record type. Upon successful completion of this statement, the first two bytes of the field contain the page group number and the last two bytes contain a db-key radix that may be used for interpreting dbkeys.

Example: The following example retrieves the page information for the DEPARTMENT record.

```
01 W_PG_INFO.
   03 W_GRP_NUM       FIXED BINARY 15,
   03 W_DBK_FORMAT    FIXED BINARY 15,

   ACCEPT PAGE_INFO RECORD (DEPARTMENT) INTO (W_PG_INFO)
```

Status Codes: After completion of the ACCEPT PAGE_INFO statement, the ERROR-STATUS field in the CA-IDMS communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
1508	The named record is not in the subschema. The program has probably invoked the wrong subschema.

6.11 ACCEPT PROCEDURE CONTROL LOCATION

The ACCEPT PROCEDURE CONTROL LOCATION statement copies the application program information block to a specified location in program variable storage. This 256-byte block is associated with a previously defined database procedure. The program information block acquires its information through the BIND PROCEDURE statement, described later in this chapter. The database procedure may have updated the information.

Only programs running under the central version, but in a different region/partition, should use the ACCEPT PROCEDURE CONTROL LOCATION statement.

For more information on the application program information block, refer to *CA-IDMS Database Administration*.

Syntax

```
►►── ACCEPT PROCEDURE (procedure-name) INTO (procedure-control-location); ──►◄
```

Parameters

procedure-name
Specifies the name of the database procedure whose application program information block will be copied into variable storage. *procedure-name* must refer to an 8-character field in variable storage.

INTO (procedure-control-location)
Specifies the fullword-aligned 256-byte location in variable storage to which the DBMS copies the application program information block.

Example: The following statement copies the application program information block used by the procedure identified in the CHECK_ALL field in main storage to the location identified as CHECK_IT in main storage:

```
ACCEPT PROCEDURE (CHECK_ALL) INTO (CHECK_IT);
```

Status codes: Upon completion of the ACCEPT PROCEDURE CONTROL LOCATION function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
1508	The subschema does not contain the named procedure.
1518	The procedure control location was not a valid address.

6.12 ACCEPT TRANSACTION STATISTICS (DC/UCF)

The ACCEPT TRANSACTION STATISTICS statement copies the contents of the transaction statistics block (TSB) to a specified location in program variable storage. Optionally, the statement can also write the TSB to the DC/UCF log file.

Syntax

```
►►── ACCEPT TRANSACTION STATISTICS ─┬───────────┬──────────────────────►
                                    ├─ WRITE ← ─┤
                                    └─ NOWRITE ─┘

►─┬─────────────────────────────────────┬─ ; ───────────────────────────►◄
  └─INTO (return-stat-data-location) ──┘
```

Parameters

WRITE/NOWRITE

Specifies whether the TSB will be written to the system log file. The default is WRITE.

INTO (return-stat-data-location)

Specifies the location to which the system copies the TSB. *Return-stat-data-location* is a fullword-aligned 388-byte field. The data copied from the TSB to *return-stat-data-location* is formatted as follows:

```
01  RETURN_STAT_DATA_LOC_V
    03  SYS_RES00      FIXED BIN (31)   RESERVED
    03  SYS_RES01      FIXED BIN (31)   RESERVED
    03  PROG_CALL      FIXED BIN (31)   # OF PROGRAMS CALLED
    03  PROG_LOAD      FIXED BIN (31)   # OF PROGRAMS LOADED
    03  TERM_READ      FIXED BIN (31)   # OF TERMINAL READS
    03  TERM_WRITE     FIXED BIN (31)   # OF TERMINAL WRITES
    03  TERM_ERROR     FIXED BIN (31)   # OF TERMINAL ERRORS
    03  STORAGE_GET    FIXED BIN (31)   # OF STORAGE GETS
    03  SCRATCH_GET    FIXED BIN (31)   # OF SCRATCH GETS
    03  SCRATCH_PUT    FIXED BIN (31)   # OF SCRATCH PUTS
    03  SCRATCH_DEL    FIXED BIN (31)   # OF SCRATCH DELETES
    03  QUEUE_GET      FIXED BIN (31)   # OF QUEUE GETS
    03  QUEUE_PUT      FIXED BIN (31)   # OF QUEUE PUTS
    03  QUEUE_DEL      FIXED BIN (31)   # OF QUEUE DELETES
    03  GET_TIME       FIXED BIN (31)   # OF GET TIMES
    03  SET_TIME       FIXED BIN (31)   # OF SET TIMES
    03  DB_CALLS       FIXED BIN (31)   # OF DATABASE CALLS
    03  MAX_STACK      FIXED BIN (31)   MAX WORDS USED IN STACK
    03  USER_TIME      FIXED BIN (31)   USER MODE TIME (10**-4 SEC)
    03  SYS_TIME       FIXED BIN (31)   SYS MODE TIME (10**-4 SEC)
    03  WAIT_TIME      FIXED BIN (31)   WAIT TIME (10**-4 SEC)
    03  RCE_USED       FIXED BIN (31)   # OF RCE'S USED
    03  RLE_USED       FIXED BIN (31)   # OF RLE'S USED
    03  DPE_USED       FIXED BIN (31)   # OF DPE'S USED
    03  STG_HI_MARK    FIXED BIN (31)   STORAGE HIGH WATER MARK
    03  FREESTG_REQ    FIXED BIN (31)   # FREE STORAGE REQUESTS
    03  SYS_SERV       FIXED BIN (31)   # SYSTEM SERVICE CALLS
    03  SYS_RES10      FIXED BIN (31)   RESERVED
    03  SYS_RES11      FIXED BIN (31)   RESERVED
    03  PAGES_READ     FIXED BIN (31)   # OF PAGES READ
    03  PAGES_WRIT     FIXED BIN (31)   # OF PAGES WRITTEN
```

```
03  PAGES_REQ      FIXED BIN (31)   # OF PAGES REQUESTED
03  CALC_NO        FIXED BIN (31)   # OF CALC RECS NO OFLOW
03  CALC_OF        FIXED BIN (31)   # OF CALC RECS OFLOW
03  VIA_NO         FIXED BIN (31)   # OF VIA RECS NO OFLOW
03  VIA_OF         FIXED BIN (31)   # OF VIA RECS OFLOW
03  RECS_REQ       FIXED BIN (31)   # OF RECS REQUESTED
03  RECS_CURR      FIXED BIN (31)   # OF RECS CURR OF RU
03  DBMS_CALLS     FIXED BIN (31)   # OF DBMS CALLS
03  FRAG_STORED    FIXED BIN (31)   # OF FRAGMENTS STORED
03  RECS_RELO      FIXED BIN (31)   # OF RECS RELOCATED
03  TOT_LOCKS      FIXED BIN (31)   TOTAL # OF LOCKS
03  SHR_LOCKS      FIXED BIN (31)   # OF SHARE LOCKS
03  NSH_LOCKS      FIXED BIN (31)   # OF NON-SHARE LOCKS
03  FREE_LOCKS     FIXED BIN (31)   # OF LOCKS FREE'D
03  SR8_SPLITS     FIXED BIN (31)   # OF SR8 SPLITS
03  SR8_SPAWNS     FIXED BIN (31)   # OF SR8 SPAWNS
03  SR8_STORED     FIXED BIN (31)   # OF SR8S STORED
03  SR8_ERASED     FIXED BIN (31)   # OF SR8S ERASED
03  SR7_STORED     FIXED BIN (31)   # OF SR7S STORED
03  SR7_ERASED     FIXED BIN (31)   # OF SR7S ERASED
03  BTREE_SRCH     FIXED BIN (31)   # OF BTREE SEARCHES
03  BTREE_LEVL     FIXED BIN (31)   # OF BTREE LEVELS SEARCHED
03  ORPHAN_ADOPT   FIXED BIN (31)   # OF ORPHANS ADOPTED
03  LVL_SRCH_BEST  FIXED BIN (15)   # LEVEL SEARCHES (BEST CASE)
03  LVL_SRCH_WORST FIXED BIN (15)   # LEVEL SEARCHES (WORST CASE)
03  RECS_UPD       FIXED BIN (31)   # OF RECS UPDATED
03  PAGE_INCACHE   FIXED BIN (31)   # OF PAGES FOUND IN CACHE
03  PAGE_INPRFET   FIXED BIN (31)   # OF PAGES FOUND IN PREFETCH
03  SYS_RES12      FIXED BIN (31)   RESERVED
03  SYS_RES13      FIXED BIN (31)   RESERVED
03  SYS_RES20      FIXED BIN (31)   RESERVED
03  SYS_RES21      FIXED BIN (31)   RESERVED
03  USER_ID        CHAR (32)        DC USER ID
03  LTERM_ID       CHAR (8)         LOGICAL TERMINAL ID
03  USER_SUPP_ID   CHAR (8)         USER-SUPPLIED ID
03  BIND_DATE      DEC FIXED (7)    DATE BIND COMMAND ISSUED
03  BIND_TIME      FIXED BIN (31)   TIME BIND COMMAND ISSUED
03  TRANSTAT_FLGS  FIXED BIN (31)   FOUR 1-BYTE FLAGS
03  SYS_RES30      FIXED BIN (31)   RESERVED
03  SYS_RES31      FIXED BIN (31)   RESERVED
03  SQL_COMMAND    FIXED BIN (31)   # OF SQL COMMANDS EXECUTED
03  SQL_FETCH      FIXED BIN (31)   # OF SQL ROWS FETCHED
03  SQL_INSERT     FIXED BIN (31)   # OF SQL ROWS INSERTED
03  SQL_UPDATE     FIXED BIN (31)   # OF SQL ROWS UPDATED
03  SQL_DELETE     FIXED BIN (31)   # OF SQL ROWS DELETED
03  SQL_SORTS      FIXED BIN (31)   # OF SQL SORTS PERFORMED
03  SQL_ROW_SORT   FIXED BIN (31)   # OF SQL ROWS SORTED
03  SQL_MIN_RSORT  FIXED BIN (31)   MINIMUM ROWS SORTED
03  SQL_MAX_RSORT  FIXED BIN (31)   MAXIMUM ROWS SORTED
03  SQL_AM_RECOMP  FIXED BIN (31)   # OF AM RECOMPILES
03  SYS_RES32      FIXED BIN (31)   RESERVED
03  SYS_RES33      FIXED BIN (31)   RESERVED
03  SYS_RES34      FIXED BIN (31)   RESERVED
03  SYS_RES35      FIXED BIN (31)   RESERVED
03  SYS_RES36      FIXED BIN (31)   RESERVED
03  SYS_RES37      FIXED BIN (31)   RESERVED
03  SYS_RES38      FIXED BIN (31)   RESERVED
03  SYS_RES39      FIXED BIN (31)   RESERVED
```

Example: The following statement returns the contents of the TSB to STATISTICS_BLOCK and writes transaction statistics to the log file:

```
ACCEPT TRANSACTION STATISTICS
   WRITE
   INTO (STATISTICS_BLOCK);
```

Status codes: Upon completion of the ACCEPT TRANSACTION STATISTICS function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
3801	The transaction statistics block has no storage available. Waiting would cause a deadlock.
3813	No transaction statistics block exists. No BIND TRANSACTION STATISTICS request was issued.
3831	Either the parameter list is invalid or no logical terminal element (LTE) is associated with the issuing task.
3850	The collection of transaction statistics or task statistics was not enabled during system generation.

6.13 ATTACH (DC/UCF)

The ATTACH statement instructs the system to initiate a new task by acquiring the necessary control blocks and storage and by adding the task to its dispatching list. The system initializes the attached task and queues it for execution. The issuing program receives control according to normal dispatching priority.

Syntax

```
►►── ATTACH TASK CODE (task-code) ─┬──────────────────────┬─┬───────────┬─ ; ─►◄
                                   └─ PRIORITY (priority) ─┘ ├─ WAIT ← ──┤
                                                             └─ NOWAIT ──┘
```

Parameters

TASK CODE <u>(task-code)</u>
Specifies the 1- to 8-character code of the task to be initiated. *Task-code* is the symbolic name of a user-defined field containing the task code or the task code itself, enclosed in single quotation marks. The referenced task code must have been defined during system generation or dynamically, by using the DCMT VARY DYNAMIC TASK command.

►► For more information about DCMT VARY DYNAMIC TASK, see *CA-IDMS System Tasks and Operator Commands*.

PRIORITY <u>(priority)</u>
Specifies the dispatching priority of the attached task. *Priority* can be the symbolic name of a user-defined fixed binary field containing the dispatching priority, or a numeric constant. Valid priorities are numeric values ranging from 000 through 240. *Priority* defaults to the priority established during system generation for the specified task code, terminal, and user.

WAIT/NOWAIT
Specifies whether the issuing task waits if a maximum task condition prevents the system from attaching the task immediately:

WAIT
Specifies that the issuing task waits until the maximum task condition no longer exists and the system can attach the specified task. WAIT is the default.

NOWAIT
Specifies that the issuing task does not wait for the system to attach the task. If you specify NOWAIT, your program should check the ERROR_STATUS field in the IDMS-DC communications block to determine whether the ATTACH request completed. If ERROR_STATUS contains the value 3711, indicating that a maximum task condition exists, then the request was not serviced and your program should perform alternative processing before reissuing the ATTACH request.

Example: The following code initiates task TASKATCH and assigns the task a dispatching priority of 199:

```
ATTACH TASK CODE (TASKATCH)
   PRIORITY (199)
   NOWAIT;
```

Status codes: Upon completion of the ATTACH function, the ERROR_STATUS field of the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
3711	The task cannot be attached because the maximum number of tasks has already been attached.
3712	The specified task code is not defined to the DC/UCF system.
3758	The task cannot be attached because the maximum number of concurrent task threads was exceeded.

6.14 BIND MAP (DC/UCF)

The BIND MAP statement identifies the location of a specified map request block (MRB) and initializes MRB fields. For each MRB used by your program, code a global BIND MAP statement. Global BIND MAP statements omit the RECORD *(record-name)* parameter. For each record defined to a map, code a record-specific BIND MAP statement. Record-specific BIND MAP statements include the RECORD *(record-name)* parameter.

Global and record-specific versions of BIND MAP: The global and record-specific versions of the BIND MAP statement function as follows:

- **Global** — The BIND MAP statement applies to the map as a whole. It initializes the entire MRB and fills in fields that apply to the map in general.
- **Record-specific** — The BIND MAP statement applies only to the named map record. It initializes the variable-storage address of the named record in the MRB.

Typically, your program issues a global BIND MAP statement for each map, followed by a BIND MAP statement for each map record used by the program.

Including BIND MAP statements automatically: You can request the DML precompiler to include global and record-specific BIND MAP statements automatically by using the INCLUDE IDMS MAP_BINDS statement (see Chapter 5, “DML Precompiler-Directive Statements” on page 5-1). This statement includes the necessary BINDS for all maps and map records defined for the program.

Altering the address for a map record: Your program can alter the storage address for a map record at any time by issuing another BIND MAP statement for that record. After the initial global bind, all map records are considered unbound. Map operations that use those records have no effect on storage. After binding a map record to a storage address with a record-specific bind, subsequent map operations use that address to access the record. To unbind a map record, issue a record-specific BIND MAP statement that specifies the TO NULL option.

Syntax

```
►►─── BIND MAP (map-name) ─┬───────────────────────────────────┬─►
                           └─ RECORD (record-name) ────────────┘

►─┬─────────────────────────────────────┬─ ; ─────────────────►◄
  └─ TO ─┬─ NULL ───────────────────┬───┘
         └─ (record-location) ──────┘
```

Parameters

map-name
Initializes the MRB associated with the named map. *Map-name* is the 1- to 8-character name of an existing map. The map version defaults to the version that you specify for the map with the DECLARE MAP statement.

RECORD (record-name)

Initializes the variable-storage address of the named record in the MRB. *Record-name* is the 1- to 32-character name of a record used by the map.

TO NULL/(record-location)

Optionally requests that the named record be unbound or specifies the address to which the record will be bound:

NULL

Requests that the DBMS not bind the named record.

record-location

Specifies the address to which the named record will be bound. *Record-location* is the symbolic name of a user-defined field that contains the address; *record-location* defaults to *record-name*. Subsequent I/O operations will use this area of storage for any operation associated with the record.

Example: The following statements bind the map EMPMAPLR and its five associated map records:

```
BIND MAP (EMPMAPLR);
BIND MAP (EMPMAPLR) RECORD (EMPLOYEE);
BIND MAP (EMPMAPLR) RECORD (DEPARTMENT);
BIND MAP (EMPMAPLR) RECORD (JOB);
BIND MAP (EMPMAPLR) RECORD (OFFICE);
BIND MAP (EMPMAPLR) RECORD (EMP-DATE-WORK-REC);
```

Status codes: Upon completion of the BIND MAP function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
1472	Insufficient memory is available for load or storage allocation.
1474	An attempt to load a module from the load library or DDLDCLOD failed.

6.15 BIND PROCEDURE

The BIND PROCEDURE statement establishes communication between your program and a DBA-written database procedure (for example, a security routine). Use this statement only in those instances in which the DBA-written procedure requires more information from your program than the DBMS provides. Such instances are unusual. Usually, you will not be aware of which procedures gain control before or after various DML functions.

You can use the BIND PROCEDURE statement in both the navigational and Logical Record Facility (LRF) environments.

Syntax

```
►►─── BIND PROCEDURE (procedure-name) TO (procedure-control-location); ──────►◄
```

Parameters

procedure-name
Specifies the name of the DBA-written database procedure for which you want to establish addressability. *Procedure-name* must refer to an 8-character field in variable storage.

TO (procedure-control-location)
Specifies the location to which the named procedure will be bound. *Procedure-control-location* is a fullword-aligned 256-byte area in variable storage.

If your program runs in a different partition than the central version, it may need to pass information to the database procedure. When the DBMS invokes the database procedure, it copies this information from the program storage area identified by *procedure-control-location* into the IDMS application program information block. The information passed is the information in *procedure-control-location* when the BIND PROCEDURE was performed; it is not the information in the program's storage at the time of the procedure call.

Example: The following statement binds the procedure with the variable name PROGCHEK to the 256-byte area PROC_CTL:

```
BIND PROCEDURE (PROGCHEK) TO (PROC_CTL);
```

Status codes: Upon completion of the BIND PROCEDURE function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.

Status code	Meaning
1400	The DBMS cannot recognize the BIND PROCEDURE statement. This code usually indicates that the IDMS-DB communications block (SUBSCHEMA_CTRL) is not aligned on a fullword boundary.
1408	The subschema does not contain the named procedure.
1418	The procedure was improperly bound to location 0.
1472	Not enough memory is available to load the database procedure dynamically.
1474	An attempt to load a module from the load library or DDLDCLOD failed.

6.16 BIND RECORD

The BIND RECORD statement establishes addressability for a record in program variable storage. In most cases, you do not have to issue individual BIND RECORD statements, since the INCLUDE IDMS SUBSCHEMA_BINDS statement generates the necessary statements as a group. (see Chapter 5, "DML Precompiler-Directive Statements" on page 5-1). Nevertheless, you can issue BIND RECORD commands separately as necessary (for example, to bind several records to the same storage location). In any case, you must establish addressability for each subschema record used by your program.

After each BIND RECORD statement, your program should perform the IDMS_STATUS routine to ensure that the statement executed successfully.

Syntax

```
►►─── BIND RECORD (record-name) ─┬──────────────────────┬─ ; ──────────►◄
                                 └─ TO (record-location) ─┘
```

Parameters

(<u>record-name</u>)
Names the record bound to a location in variable storage. The location corresponds to the record description copied into the program. *Record-name* must specify a record included in the subschema.

TO (<u>record-location</u>)
Optionally allows you to bind the record to a specific location. The data defined in *record-location* must be identical in length to the data defined in *record-name*.

Note: Be careful when using the TO *(record-location)* option. Source-object mismapping can result from improper use. If your program contains more than one copy of a given database record description, you must be sure to bind the proper record description at the proper time.

Example: The following statement binds the EMPLOYEE record:

```
BIND RECORD (EMPLOYEE);
```

Status codes: Upon completion of the BIND RECORD function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
1400	The DBMS cannot recognize the BIND RECORD statement. This code usually indicates that the IDMS-DB communications block (SUBSCHEMA_CTRL) is not aligned on a fullword boundary.

Status code	Meaning
1408	The subschema does not contain the named record. Your program probably invoked the wrong subschema.
1418	The record was improperly bound to location 0.
1472	Insufficient memory is available to load a database procedure dynamically.
1474	An attempt to load a module from the load library or DDLDCLOD failed.

6.17 BIND RUN_UNIT

The BIND RUN_UNIT statement:

- Establishes a run unit for accessing the database
- Identifies the location of the IDMS-DB communications block being used
- Names the subschema to be loaded for the run unit
- Names the node under which the run unit will execute
- Identifies the database to be accessed
- Identifies the dictionary in which a subschema resides
- Identifies the node that controls the dictionary

BIND RUN_UNIT must be the first functional DML call passed to the DBMS at execution time. BIND RUN_UNIT must logically precede all other DML statements (for example, BIND RECORD, READY, FIND) in your program.

When you don't need BIND RUN_UNIT: If you use the INCLUDE IDMS SUBSCHEMA_BINDS statement (see Chapter 5, "DML Precompiler-Directive Statements" on page 5-1) in your program, you do not need the BIND RUN_UNIT statement. INCLUDE IDMS SUBSCHEMA_BINDS automatically invokes the necessary binds.

Program registration: Some sites require program registration, that is, they require all programs to be registered in the dictionary before compilation. If your site requires program registration, your program must initialize the PROGRAM_NAME field of the IDMS communications block either automatically or manually:

- **Automatically** — A PL/I assignment statement automatically generated by INCLUDE IDMS SUBSCHEMA_BINDS moves the program name to the PROGRAM_NAME field.
- **Manually** — You code a PL/I assignment statement prior to the BIND RUN_UNIT statement. For example:

```
PROGRAM_NAME = 'EMPDISP';
```

You can use the BIND RUN_UNIT statement in both the navigational and Logical Record Facility (LRF) environments.

Syntax

```
►►── BIND RUN_UNIT ─┬──────────────────────────────┬──────────────────────────►
                    └─ SUBSCHEMA (subschema-name) ─┘

►─┬─────────────────────────────────┬───────────────────────────────────────────►
  └─┬─ DBNODE ───┬─ (nodename) ─────┘
    └─ NODENAME ─┘

►─┬──────────────────────────┬─┬──────────────────────────┬──────────────────────►
  └─ DBNAME (database-name) ─┘ └─ DICTNODE (nodename) ─────┘

►─┬───────────────────────────────┬─ ; ──────────────────────────────────────────►◄
  └─ DICTNAME (dictionary-name) ──┘
```

Parameters

SUBSCHEMA (<u>subschema-name</u>)
Identifies a subschema view other than that specified in the DECLARE SUBSCHEMA statement. *Subschema-name* must be the 1- to 8-character name of a subschema.

Note: You should use the SUBSCHEMA *subschema-name* option carefully. Improper use can lead to mismappings between the named subschema and record descriptions in variable storage.

DBNODE/NODENAME (<u>nodename</u>)
Specifies the node where the database resides. *Nodename* is either the symbolic name of a user-defined 8-character field in variable storage or the node name itself, enclosed in single quotation marks. The keywords DBNODE and NODENAME are synonymous.

DBNAME (<u>database-name</u>)
Names the database to be accessed by the run unit. *Database-name* is either the symbolic name of a user-defined 8-character field in variable storage, or the database name itself enclosed in single quotation marks.

DICTNODE (<u>nodename</u>)
Names the node that controls the data dictionary where the subschema resides. *Nodename* is either the symbolic name of a user-defined 8-character field in variable storage, or the nodename itself enclosed in single quotation marks.

DICTNAME (<u>dictionary-name</u>)
Names the dictionary where the subschema resides. *Dictionary-name* is either the symbolic name of a user-defined 8-character field in variable storage, or the dictionary name itself enclosed in single quotation marks.

Note: Specifying DBNODE, DBNAME, DICTNODE, and DICTNAME as BIND RUN_UNIT parameters overrides any corresponding parameters set using the system DCUF SET statement (online) or the SYSIDMS job stream parameters (batch). For information on DCUF SET, refer to *CA-IDMS System Tasks and Operator Commands*. For information on SYSIDMS, refer to *CA-IDMS Database Administration*.

Example: The following example illustrates how a batch program accesses a subschema, EMPSS01, stored in dictionary PRODICT1 at node DEVT. The run unit accesses database PRODDB1 at the same node.

```
BIND RUN_UNIT SUBSCHEMA (EMPSS01) NODENAME (DEVT)
    DBNAME (PRODDB1) DICTNODE (DEVT) DICTNAME (PRODICT1);
```

Status codes: Upon completion of the BIND RUN_UNIT function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request was serviced successfully.
1400	The DBMS cannot recognize the BIND RUN_UNIT statement. This code usually indicates that the IDMS-DB communications block (SUBSCHEMA_CTRL) is not aligned on a fullword boundary.
1467	The subschema invoked does not match the subschema object tables.
1469	The run unit is not bound to the DBMS. This code indicates that the central version is not active, that the central version is not accepting new run units, or that the run unit's connection to the central version is broken due to timeout or other factors, as noted on the CV log.
1470	A journal file will not open (local mode only); the most probable cause is that the JCL doesn't correctly specify the journal file.
1472	The available memory is insufficient to load a subschema or database procedure dynamically.
1473	The central version is not accepting new run units.
1474	The subschema was not found in the dictionary load area or in the load library.
1477	The run unit was already bound.
1480	The node specified in the DBNODE clause is not active or was disabled from the system generation configuration.
1481	IDMS does not know the database specified in the DBNAME clause.
1482	The named subschema is not valid under the database specified in the DBNAME clause.
1483	The available memory is insufficient to allocate native VSAM work areas.

6.18 BIND TASK (DC/UCF)

The BIND TASK statement initiates a system task when the operating mode is DC_BATCH. This statement establishes communication with the DC/UCF system and, if accessing system queues, allocates a packet-data movement buffer to contain the queue data. Once a task is started, the program can issue any number of consecutive BIND-READY-FINISH sequences.

Syntax

```
►►── BIND TASK ──────────────────────────────────────────────────────────►

►─┬──────────────────────────┬─ ; ──────────────────────────────────────►◄
  └─ NODENAME (nodename) ────┘
```

Parameters

NODENAME <u>(nodename)</u>
Specifies the 1- to 8-character name of the node to which the task will be bound. *Nodename* is either the symbolic name of a user-defined field that contains the node name or the node name itself enclosed in single quotation marks. The specified node name must match the node named in the DDS statement at system generation.

Example: The following statement establishes communication with a DC/UCF system:

```
BIND TASK;
```

Status codes: Upon completion of the BIND TASK function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.

6.19 BIND TRANSACTION STATISTICS (DC/UCF)

The BIND TRANSACTION STATISTICS statement defines the beginning of a transaction for the purposes of collecting transaction statistics. The system allocates a block of storage in which to accumulate these statistics. Because this block is owned by the logical terminal associated with the current task, the BIND TRANSACTION STATISTICS statement cannot be used with nonterminal tasks.

Note: If a transaction statistics block (TSB) is already allocated for the logical terminal associated with the current task, the BIND request clears the block and writes any previously accumulated transaction statistics to the log file.

When a BIND TRANSACTION STATISTICS request is issued, the system assigns the transaction a 40-character identifier; the first 32 characters are the identifier of the signed-on user (if any) and the last eight characters are the identifier of the logical terminal associated with the current task.

Syntax

```
►►─── BIND TRANSACTION STATISTICS; ──────────────────────────────►◄
```

Example: The following example illustrates the BIND TRANSACTION STATISTICS statement:

```
BIND TRANSACTION STATISTICS;
```

Status codes: Upon completion of the BIND TRANSACTION STATISTICS function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully; any existing transaction statistics block was written to the log file before being cleared.
3801	Storage for the transaction statistics block is not available; to wait would cause a deadlock.
3810	A new transaction statistics block has been allocated.
3831	Either the parameter list is invalid or no logical terminal element (LTE) is associated with the issuing task.
3850	The collection of transaction statistics or task statistics has not been enabled during system generation.

6.20 CHANGE PRIORITY (DC/UCF)

The CHANGE PRIORITY statement changes the dispatching priority of the issuing task. The new dispatching priority applies only to the current execution of the task. CHANGE PRIORITY does not relinquish control to another task and cannot be used to alter the priority of other tasks.

Syntax

```
►►── CHANGE PRIORITY TO (priority); ──────────────────────────────────►◄
```

Parameter

priority

Specifies a new dispatching priority for the issuing task. *Priority* is either the symbolic name of a user-defined field that contains the priority value, or the value itself expressed as a numeric constant in the range 0 through 240.

Example: The following example changes the dispatching priority of the issuing task to the value contained in the PRIORITY_210 field:

```
CHANGE PRIORITY TO (PRIORITY_210);
```

Status codes: Upon completion of the CHANGE PRIORITY function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.

6.21 CHECK TERMINAL (DC/UCF)

The CHECK TERMINAL statement tests whether a previously issued asynchronous I/O operation is complete. If a READ TERMINAL, WRITE TERMINAL, or WRITE THEN READ TERMINAL request specifies the NOWAIT option, the program must issue a CHECK TERMINAL request before specifying any other I/O operation. If the I/O operation is not complete, the system suspends task execution. When the I/O operation is complete, the task resumes execution according to its established dispatching priority.

Syntax

```
►►── CHECK TERMINAL; ──────────────────────────────────────────────►◄
```

Status codes: Upon completion of the CHECK TERMINAL function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4519	The input area specified for the return of data is too small; the returned data has been truncated to fit the available space.
4525	The output operation has been interrupted; the terminal operator has pressed ATTENTION or BREAK.
4526	A logical error (for example, an invalid control character) has been encountered in the output data stream.
4527	A permanent I/O error has occurred during processing.
4528	The dial-up line for the terminal being used has been disconnected.
4531	The terminal request block (TRB) contains an invalid field, indicating a possible error in the program's parameters.
4539	The terminal device associated with the issuing task is out of service.

6.22 COMMIT

The COMMIT statement requests that CA-IDMS/DB write a checkpoint to the journal file to designate the start or end of specific database, scratch, or queue record access activities associated with the issuing run unit or task. COMMIT simulates a FINISH-BIND-READY sequence without relinquishing control of database resources.

You can use the COMMIT statement in both the navigational and Logical Record Facility (LRF) environments.

Currency: Use of the ALL parameter with COMMIT (COMMIT ALL) sets all currencies to null.

Syntax

```
►►─── COMMIT ─┬──────────┬─┬─────────┬─ ; ──────────────────────►◄
              └─ TASK ───┘ └─ (ALL) ─┘
```

Parameters

TASK

Establishes checkpoints for all data areas associated with all run units, scratch activities, queue activities, and print activities initiated by the issuing task.

Note: The TASK parameter applies to CA-IDMS/DC only.

(ALL)

Releases all locks held on records in database, scratch, and queue areas associated with the issuing task or run unit, and sets all currencies to null.

Example: The following statement writes a commit checkpoint to the journal file:

```
COMMIT;
```

Status codes: Upon completion of the COMMIT function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
5031	The specified request is invalid; the program may contain a logic error.
5096	Too many run units are currently active; check the system log for details.
5097	An invalid status has been received from DBIO/DBMS; check the system log for details.

6.23 CONNECT

The CONNECT statement establishes a record occurrence as a member of a set occurrence. The specified record must be defined as an optional automatic, optional manual, or mandatory manual member of the set.

Native VSAM users: The CONNECT statement is not valid since all sets in native VSAM data sets must be defined as mandatory automatic.

Before executing the CONNECT statement, satisfy these conditions:

- Ready all areas affected either explicitly or implicitly by the CONNECT statement in one of the update usage modes (see 6.61, "READY" on page 6-164 later in this chapter).
- Establish the specified record as current of its record type.
- Establish the occurrence of the set into which the specified record will be connected. The current record of set determines the set occurrence and, if set order is NEXT or PRIOR, the position at which the specified record will be connected within the set.

Currency: Following successful execution of a CONNECT statement, the specified record is current of run unit, its record type, its area, and all sets in which it currently participates.

Syntax

```
►►── CONNECT RECORD (record-name) SET (set-name); ──────────────────►◄
```

Parameters

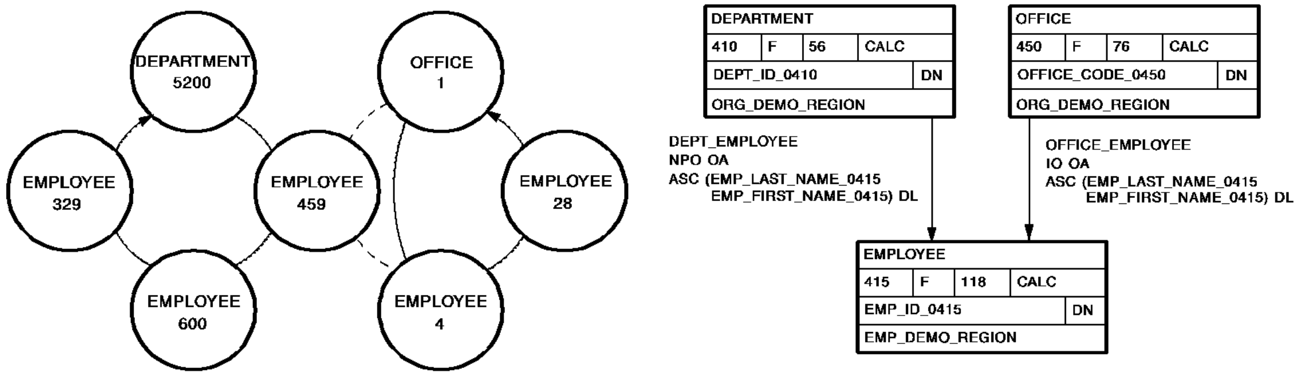
RECORD (record-name)
Specifies the record type to be connected. *Record-name* must be a record included in the subschema and must be defined as an optional automatic, optional manual, or mandatory manual member of the set to which it is being connected.

SET (set-name)
Specifies the set to which the member record is to be connected. *Set-name* must be a set included in the subschema. The record is connected to the set in accordance with the ordering rules defined for that set in the schema.

Example: The following statement connects the current EMPLOYEE record to the current occurrence of the OFFICE_EMPLOYEE set:

```
CONNECT RECORD (EMPLOYEE) SET (OFFICE_EMPLOYEE);
```

The following figure illustrates the steps required to connect an EMPLOYEE record to an occurrence of the OFFICE_EMPLOYEE set. To connect EMPLOYEE 459 to OFFICE 1 in the OFFICE_EMPLOYEE set, establish EMPLOYEE 459 as current of record type, locate the proper occurrence of the OFFICE record, and issue the CONNECT command.



	CURRENCIES RUN UNIT, RECORD, SET, AREA							
	RUN UNIT	DEPARTMENT	EMPLOYEE	OFFICE	DEPT_EMPLOYEE	OFFICE_EMPLOYEE	ORG_DEMO_REGION	EMP_DEMO_REGION
DEPT_ID = 5200 FIND CALC RECORD (DEPARTMENT);	5200	5200			5200		5200	
OBTAIN FIRST RECORD (EMPLOYEE), SET (DEPT_EMPLOYEE);	459	5200	459		459		5200	459
OFFICE_CODE = 1; FIND CALC RECORD (OFFICE);	1	5200	459	1	459	1	1	459
CONNECT RECORD (EMPLOYEE) SET (OFFICE_EMPLOYEE).	459	5200	459	1	459	459	1	459

Status codes: Upon completion of the CONNECT function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0705	The CONNECT would violate a duplicates-not-allowed option.
0706	Currency has not been established for the named record or set.
0708	The named record is not in the subschema. The program has probably invoked the wrong subschema.
0709	The named record's area has not been readied in one of the update usage modes.

Status code	Meaning
0710	The subschema specifies an access restriction that prohibits connecting the named record in the named set.
0714	The CONNECT statement cannot be executed because the named record has been defined as a mandatory automatic member of the set.
0716	The record cannot be connected to a set in which it is already a member.
0721	An area other than the area of the named record has been readied with an incorrect usage mode.
0725	Currency has not been established for the named set type.

6.24 DC RETURN (DC/UCF)

The DC RETURN statement returns control to a program at the next higher level within a task. Additionally, you can use the DC RETURN statement to specify:

- The next task to be initiated on the same terminal
- Recovery procedures for abend routines established by SET ABEND EXIT (STAE) functions
- The action to be taken by the system if the terminal operator fails to initiate the next task

Control returns to the program or system: Following a DC RETURN request, control returns to the program at the next higher level within the task. If the issuing program is the highest level program, control returns to the system. Any DC RETURN statement can include a NEXT TASK CODE option to specify the next task to be initiated by the system. However, the position of the issuing program within the task governs whether the specified task will, in fact, receive control.

When the system receives control from the highest level program that issued a DC RETURN NEXT TASK CODE request, the specified task is executed immediately if the specified task code has been assigned the NOINPUT attribute during system generation; if the task code was assigned the INPUT attribute, the task executes only when the terminal operator presses an attention identifier (AID) key. Typical AID keys include all PA and PF keys, ENTER, and CLEAR.

Syntax

```
►►── DC RETURN ─┬──────────────────────────────────────┬──────────────────────────►
                └─ NEXT TASK CODE (next-task-code) ─┘

►─┬──────────────────────────────────────────────────────┬─ ; ─────────────────►◄
  └─ TIMEOUT ─↓─┬─ INTERVAL (timeout-interval) ─┬─┴─┘
                └─ PROGRAM (timeout-program) ──┘
```

Parameters

NEXT TASK CODE <u>(next-task-code)</u>

Specifies the 1- to 8-character code associated with a task to be initiated on the same terminal. *Next-task-code* is either the symbolic name of a user-defined field that contains the task code or the task code itself enclosed in single quotation marks. The specified task code must be defined to the system under which the task is running, either during system generation or at runtime, by using a DCMT VARY DYNAMIC TASK command. For more information about DCMT VARY DYNAMIC TASK, refer to *CA-IDMS System Tasks and Operator Commands*.

TIMEOUT

Specifies the action the system is to take if the terminal operator fails to enter data required to initiate a task. This parameter overrides resource timeout interval and program specifications established during system generation.

INTERVAL **<u>(timeout-interval)</u>**
Specifies the time, in seconds, that can elapse before the system releases the resources held by the terminal on which the task is executing.
Timeout-interval is either the symbolic name for a user-defined FIXED BINARY(31) field that contains the timeout interval or the interval itself expressed as a numeric constant.

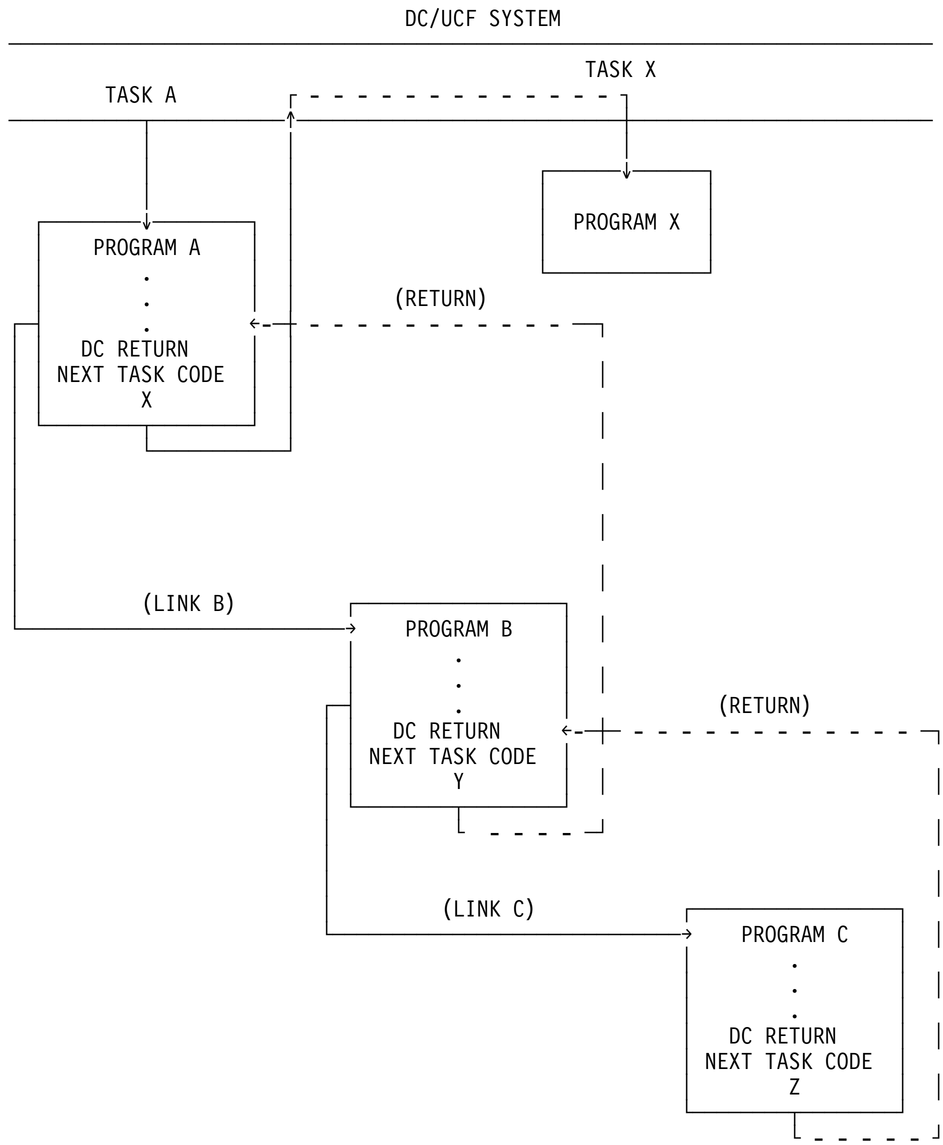
PROGRAM **<u>(timeout-program)</u>**
Specifies the 1- to 8-character name of the program to be invoked when the specified timeout interval has been reached. This program handles and releases resources held by the terminal on which the task was executing.
Timeout-program is either the symbolic name of a user-defined field that contains the program name or the name itself enclosed in single quotation marks. The specified program must be defined to the system either during system generation or at runtime by using a DCMT VARY DYNAMIC PROGRAM command. For more information on DCMT VARY DYNAMIC PROGRAM, refer to *CA-IDMS System Tasks and Operator Commands*.

Example: The following statement illustrates the use of DC RETURN. The task code associated with MENU_TASK_CODE, if defined with the INPUT parameter, will be invoked the next time the terminal operator presses an attention identifier (AID) key; if MENU_TASK_CODE is defined with the NOINPUT parameter, it will be invoked immediately.

```
DC RETURN
   NEXT TASK CODE (MENU_TASK_CODE);
```

The following figure illustrates how the system executes a task when DC RETURN statements within three programs specify the NEXT TASK CODE option.

In DC RETURN Processing Task A invokes program A. Program A links to program B, which in turn links to program C. Program C issues a DC RETURN NEXT TASK CODE ('Z') request; control returns to program B. Program B contains a DC RETURN NEXT TASK CODE ('Y') request, which takes precedence over program C's DC RETURN specification. Control returns to program A, which issues a DC RETURN NEXT TASK CODE ('X') request. Because program A is at the highest level in the task, task X will be invoked.



Status codes: Because control is returned to the next-higher level, there is no need to check the ERROR_STATUS field.

6.25 DELETE QUEUE (DC/UCF)

The DELETE QUEUE statement deletes all or part of a queue. If only one queue record is deleted, the system maintains currency within the queue by saving the next and prior currencies of the deleted record.

Syntax

```
►►── DELETE QUEUE ─┬──────────────────┬─┬─ CURRENT ← ─┬─ ; ──────────────►
                   └─ ID (queue-id) ──┘ └─ ALL ───────┘
```

Parameters

ID (queue-id)
Specifies the 1- to 16-character ID of the queue that contains the record to be deleted. *Queue-id* is either the symbolic name of a user-defined field that contains the ID or the ID itself enclosed in single quotation marks. If the queue ID is not specified, a blank ID is assumed.

CURRENT
Deletes the current record of the queue associated with the requesting task. CURRENT is the default.

ALL
Deletes all records in the queue and the queue header id.

Example: The following statement deletes the current record in the RES_Q queue:

```
DELETE QUEUE
   ID ('RES_Q')
   CURRENT;
```

Status codes: Upon completion of the DELETE QUEUE function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4404	The requested queue header record cannot be found.
4405	The requested queue record cannot be found.
4406	No resource control element (RCE) exists for the queue record, indicating that currency has not been established.
4407	An I/O error has occurred during processing, or the queue upper limit has been reached.
4431	The parameter list is invalid.

6.26 DELETE SCRATCH (DC/UCF)

The DELETE SCRATCH statement deletes one scratch record or all records in the scratch area.

Syntax

```
►►── DELETE SCRATCH─┬──────────────────────────────┬──────────────────►
                    └─ AREA ID (scratch-area-id ) ─┘

►─┬───────────────────────────────────┬───────────────────────────────►
  ├─ CURRENT ─────────────────────────┤
  ├─ FIRST ───────────────────────────┤
  ├─ LAST ────────────────────────────┤
  ├─ NEXT ────────────────────────────┤
  ├─ PRIOR ───────────────────────────┤
  ├─ ALL ─────────────────────────────┤
  └─ RECORD ID (scratch-record-id) ───┘

►─┬────────────────────────────────────────────────────┬─ ; ─────────►◄
  └─ RETURN RECORD ID INTO (return-scratch-record-id) ─┘
```

Parameters

AREA ID <u>(scratch-area-id)</u>
Specifies the 1- to 8-character ID of the scratch area associated with the scratch records being deleted. *Scratch-area-id* is either the symbolic name of a user-defined field that contains the scratch area ID or the ID itself enclosed in single quotation marks. If the AREA ID parameter is not specified, the system assumes an area ID of 8 blanks.

CURRENT
Deletes the current record in the specified scratch area (that is, that record most recently referenced by another scratch function). CURRENT is the default.

FIRST
Deletes the first record in the specified scratch area.

LAST
Deletes the last record in the specified scratch area.

NEXT
Deletes the next record in the specified scratch area.

PRIOR
Deletes the prior record in the specified scratch area.

ALL
Deletes all records in the specified scratch area.

RECORD ID <u>(scratch-record-id)</u>
Deletes the record identified by *scratch-record-id*. *Scratch-record-id* is the symbolic name of a user-defined field that contains the ID.

RETURN RECORD ID INTO (return-scratch-record-id)
Specifies the location in the program to which the system will return the ID of the last record deleted by means of the DELETE SCRATCH function. *Return-scratch-record-id* is the symbolic name of a user-defined 4-byte field.

Example: The following statement deletes the scratch record that is prior to the current scratch record and returns the ID of the deleted record to the SCR_REC_ID field:

```
DELETE SCRATCH
   PRIOR
   RETURN RECORD ID INTO (SCR_REC_ID);
```

Status codes: Upon completion of the DELETE SCRATCH function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4303	The requested scratch area ID cannot be found.
4305	The requested scratch record ID cannot be found.
4307	An I/O error has occurred during processing.
4331	The parameter list is invalid.

6.27 DELETE TABLE (DC/UCF)

The DELETE TABLE statement notifies the system that the issuing task has finished using a table that has been loaded into the program pool by using the LOAD TABLE function. DELETE TABLE does not physically delete reusable tables from the program pool; rather, it decrements the in-use count maintained by the DC/UCF system. An in-use count of 0 signals to the system that the space occupied by the table can be reused.

Syntax

```
►►── DELETE TABLE FROM (table-location-pointer); ──────────────────►◄
```

Parameter

table-location-pointer
Specifies a table location where the in-use count maintained by the system is to be decremented. *Table-location-pointer* specifies the variable-storage pointer location that was set when the table was loaded via a LOAD TABLE request.

Example: The following example releases a previously loaded table from the location in variable storage identified by RATE_TABLE_PTR:

```
DELETE TABLE FROM (RATE_TABLE_PTR);
```

Status codes: Upon completion of the DELETE TABLE function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
3433	The specified table was not loaded by the task.

6.28 DEQUEUE (DC/UCF)

The DEQUEUE statement releases resources acquired by the issuing task with an ENQUEUE request. Acquired resources not released explicitly with a DEQUEUE request are released automatically at task termination.

Syntax

```
►►─── DEQUEUE ─┬─── ALL ──────────────────────────────────────────────────┬─ ; ──►◄
               │    ┌──────────────────────────────────────────────────┐  │
               └─↓─ NAME (resource-id) LENGTH (resource-id-length) ────┴──┘
```

Parameters

ALL
Releases all resources acquired by the issuing task by means of ENQUEUE requests.

NAME (resource-id)
Specifies the resources to be dequeued and supplies the length of each resource: *Resource-id* is the symbolic name of a user-defined field that contains the 1- to 255-character resource ID. Multiple NAME parameters must be separated by at least one blank.

LENGTH (resource-id-length)
Specifies either the symbolic name of a user-defined FIXED BINARY(31) field that contains the length of the resource ID, or the length itself expressed as a numeric constant.

Example: The following statement releases all the resources enqueued by the issuing task:

```
DEQUEUE NAME (PAYROLL_LOCK)
        LENGTH (16);
```

Status codes: Upon completion of the DEQUEUE function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
3913	At least one resource ID cannot be found; all resources that were located have been dequeued.
3931	The parameter list is invalid.

6.29 DISCONNECT

The DISCONNECT statement cancels the current membership of a record occurrence in a set occurrence. The named record must be defined as an *optional* member of the named set.

Native VSAM users: The DISCONNECT statement is not valid since all sets in native VSAM data sets must be defined as mandatory automatic.

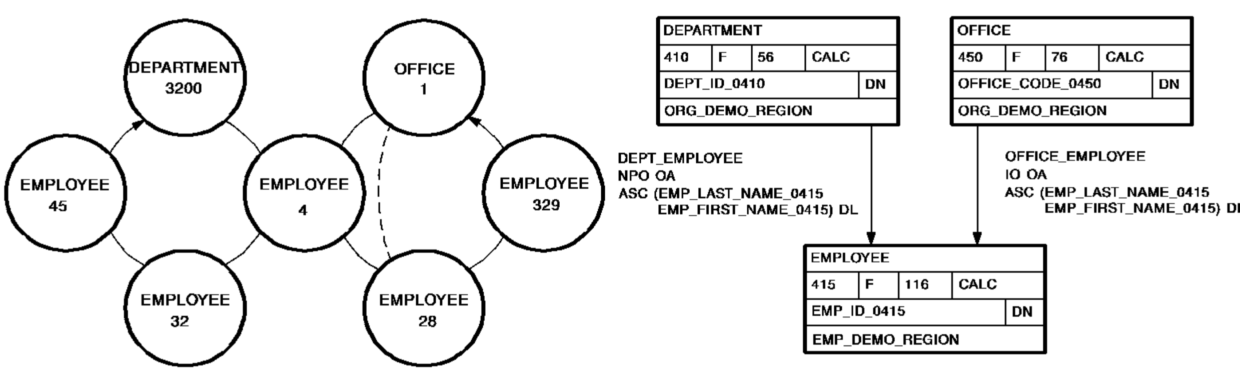
Before executing the DISCONNECT statement, satisfy the following conditions:

- Ready all areas affected either explicitly or implicitly by the DISCONNECT statement with one of the three update usage modes (see 6.61, “READY” on page 6-164, later in this chapter).
- Establish the named record as current of its record type.
- Make sure that the named record currently participates as a member in an occurrence of the named set.

Following successful execution of the DISCONNECT statement, the named record can no longer be accessed through the set for which membership was canceled. The disconnected record can still be accessed either by means of a complete scan of the area in which it participates or directly through its db-key, if known. A disconnected record can also be accessed either through any other sets in which it participates as a member or if it has a location mode of CALC.

Currency: A successfully executed DISCONNECT statement nullifies currency in the specified set. However, next, prior, and owner of set are maintained, enabling continued access within the set. The disconnected record is current of run unit, its record type, its area, and any other sets in which it participates. The following figure illustrates the steps required to disconnect an EMPLOYEE record from an occurrence of the OFFICE_EMPLOYEE set.

To disconnect EMPLOYEE 4 from OFFICE 1 of the OFFICE_EMPLOYEE set, enter the database on OFFICE 1, establish EMPLOYEE 4 as current of the EMPLOYEE record type, and disconnect it from the OFFICE_EMPLOYEE set.



	CURRENCIES RUN UNIT, RECORD, SET, AREA							
	RUN UNIT	EMPLOYEE	DEPARTMENT	OFFICE	DEPT_EMPLOYEE	OFFICE_EMPLOYEE	ORG_DEMO_REGION	EMP_DEMO_REGION
OFFICE_CODE = 1; FIND CALC RECORD (OFFICE);	1			1		1	1	
FIND FIRST RECORD (EMPLOYEE) SET (OFFICE_EMPLOYEE);	4	4		1	4	4	1	4
DISCONNECT RECORD (EMPLOYEE) SET (OFFICE_EMPLOYEE);	4	4		1	4	NPO	1	4

Syntax

```
►►── DISCONNECT RECORD (record-name) SET (set-name); ──────────────►◄
```

Parameters

RECORD (record-name)
Specifies the record type to be disconnected. *Record-name* must be a record included in the subschema and must be defined as an optional member of the specified set.

SET (set-name)
Specifies the set from which the named record will be disconnected. *Set-name* must be a set included in the subschema.

Example: The following statement disconnects the current EMPLOYEE record from the OFFICE_EMPLOYEE set:

```
DISCONNECT RECORD (EMPLOYEE) SET (OFFICE_EMPLOYEE);
```

Status codes: Upon completion of the DISCONNECT function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
1106	Currency has not been established for the named record.
1108	The named record is not in the subschema. The program has probably invoked the wrong subschema.
1109	The named record's area has not been readied in one of the update usage modes.

Status code	Meaning
1110	The subschema specifies an access restriction that prohibits use of the DISCONNECT statement.
1115	The DISCONNECT statement cannot be executed because the named record has been defined as a mandatory member of the set.
1121	An area other than the area that contains the named record has been readied with an incorrect usage mode.
1122	The named record is not currently a member of the specified set.

6.30 END LINE TERMINAL SESSION (DC/UCF)

The END LINE TERMINAL SESSION statement terminates the current line-mode I/O session. All output data lines that remain in the current buffer and all pages queued for asynchronous I/O operations are deleted.

Syntax

```
►►─── END LINE TERMINAL session ; ─────────────────────────────────────►◄
```

Example: The following statement terminates a line mode I/O session:

```
END LINE TERMINAL SESSION;
```

Status codes: There are no codes associated with the END LINE TERMINAL SESSION command.

6.31 END TRANSACTION STATISTICS (DC/UCF)

The END TRANSACTION STATISTICS statement defines the end of a transaction. The transaction typically ends when the issuing task terminates. Optionally, END TRANSACTION STATISTICS can be used to write the transaction statistics block (TSB) to the system log file and to return the TSB to a preallocated location in variable storage. The system returns a copy of the TSB to program variable storage.

Syntax

```
►►─── END TRANSACTION STATISTICS ─┬────────────┬──────────────────────────►
                                   ├─ WRITE ← ─┤
                                   └─ NOWRITE ─┘

►─┬─────────────────────────────────────────┬─ ; ──────────────────────────►◄
  └─ INTO (return-stat-data-location) ─┘
```

Parameters

WRITE/NOWRITE
Indicates whether the TSB will be written to the system log file when the task terminates. The default is WRITE.

INTO (return-stat-data-location)
Specifies the location in program variable storage into which the system will return the TSB. *Return-stat-data-location* is the symbolic name of a user-defined field.

Example: The following statement ends a transaction, writes statistics to the log file, and returns a copy of the TSB to the STATISTICS_BLOCK field:

```
END TRANSACTION STATISTICS
   WRITE
   INTO (STATISTICS_BLOCK);
```

Status codes: Upon completion of the END TRANSACTION STATISTICS function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
3801	Storage for the transaction statistics block is not available; to wait would cause a deadlock.
3813	No transaction statistics block exists; a BIND TRANSACTION STATISTICS request has not been issued.
3831	Either the parameter list is invalid or no logical terminal element (LTE) is associated with the issuing task.
3850	The collection of transaction statistics or task statistics has not been enabled during system generation.

6.32 ENDPAGE (DC/UCF)

The ENDPAGE statement terminates a map paging session, clears the scratch record for the session, and clears the map paging options for the completed session. A STARTPAGE/ENDPAGE pair encloses commands that handle a pageable map at runtime. The STARTPAGE command is discussed later in this chapter.

Syntax

```
►►── ENDPAGE session ; ─────────────────────────────────────────►◄
```

Example: The following statement ends a map paging session:

```
ENDPAGE SESSION;
```

Status codes: Upon completion of the ENDPAGE function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.

6.33 ENQUEUE (DC/UCF)

The ENQUEUE statement acquires or tests the availability of a resource or list of resources. Resources are defined during installation and system generation and typically include storage areas, common routines, queues, and processor time.

An enqueued resource can be exclusive or shared:

- Exclusive — The resource is owned exclusively by the issuing task and is not available to any other tasks. The system prohibits other tasks from obtaining resources that have been ENQUEUED exclusively.

  Note: An exclusive ENQUEUE request prohibits another task from enqueueing a resource by name; however, it does not prohibit the use of the resource by another task. Therefore, to effect true resource protection, you must enqueue and dequeue resources consistently.

- Shared — The resource is available to all tasks. The system allows other tasks to issue nonexclusive ENQUEUE requests for the resources, permitting the resources to be shared.

Syntax

```
►►── ENQUEUE ─┬───────────┬──────────────────────────────────────────────────►
              ├─ WAIT ← ──┤
              ├─ NOWAIT ──┤
              └─ TEST ────┘

  ┌──────────────────────────────────────────────────────────────────────┐
►─↓─ NAME (resource-id) LENGTH (resource-id-length) ─┬─────────────────┬─┴─ ; ─►◄
                                                      ├─ EXCLUSIVE ← ──┤
                                                      └─ SHARED ───────┘
```

Parameters

WAIT
Specifies that the system is to wait for all resources to be freed if it cannot service the request immediately. WAIT is the default.

NOWAIT
Specifies that the system is not to wait to acquire resources that are not currently available. If NOWAIT is specified, the program should check the ERROR_STATUS field in the IDMS-DC communications block to determine if the function has been completed. If the ERROR_STATUS value is 3901, indicating that a resource could not be obtained immediately, the request has not been serviced and the program should perform alternative processing before reissuing the NOWAIT request.

TEST
Tests the availability of the specified resources. If TEST is specified, the program should check the ERROR_STATUS field in the IDMS-DC communications block to determine the outcome of the test.

NAME (resource-id)
Names the 1- to 255-character ID associated with the resource. *Resource-id* must be a user-defined field that contains the resource ID. The specified resource ID must be the name of a resource defined to the system. Any resource name can be specified, provided that all programs that access the resource use the same name. Multiple NAME parameters must be separated by at least one blank.

LENGTH (resource-id-length)
Specifies the symbolic name of either a user-defined FIXED BINARY(31) field that contains the length of the resource ID or the length itself expressed as a numeric constant.

EXCLUSIVE/SHARED
Assigns the exclusive or shared attribute to the named resource. The default attribute is EXCLUSIVE.

Example: The following statement enqueues the CODE_VALUE and PAYROLL_LOCK resources. CODE_VALUE is reserved for exclusive use by the issuing task; PAYROLL_LOCK can be shared.

```
ENQUEUE
   WAIT
   NAME (CODE_VALUE) LENGTH (10)
   NAME (PAYROLL_LOCK) LENGTH (16) SHARED;
```

The following statement tests the availability of the resource whose identifier is contained in the RESOURCE_NAME field:

```
ENQUEUE
   TEST
   NAME (RESOURCE_NAME) LENGTH (RESOURCE_NAME_LENGTH);
```

Status codes: Upon completion of an ENQUEUE function to *acquire* resources, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
3901	At least one of the requested resources cannot be enqueued immediately; to wait would cause a deadlock. No new resources have been acquired.
3908	At least one of the requested exclusive resources is currently owned by another task. No new resources have been acquired.
3931	The parameter list is invalid.

Upon completion of an ENQUEUE function to *test* resources, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	All requested resources are available.
3908	At least one of the tested resources is already owned by another task.
3909	At least one of the tested resources is not yet owned by another task and is available to the issuing task.
3931	The parameter list is invalid.

6.34 ERASE

The ERASE statement performs the following functions:

- Disconnects the specified record from all set occurrences in which it participates as a member and logically or physically deletes the record from the database
- Optionally erases all records that are mandatory members of set occurrences owned by the specified record
- Optionally disconnects or erases all records that are optional members of set occurrences owned by the specified record

ERASE is a two-step procedure that first cancels the existing membership of the named record in specific set occurrences and then releases for reuse the space occupied by the named record and its db-key. Erased records are unavailable for further processing by any DML statement.

Before executing the ERASE statement, satisfy the following conditions:

- Ready all areas that are affected either implicitly or explicitly in one of the update usage modes (see 6.61, “READY” on page 6-164 later in this chapter).
- Include and ready in an update usage mode all sets in which the specified record participates as a member.
- Include in the subschema all sets in which the specified record participates as owner either directly or indirectly (for example, as owner of a set with a member that is owner of another set) and all member record types in those sets.
- Include in the subschema all records that participate either implicitly or explicitly as owners.
- Establish the specified record as current of run unit.

Currency: Following successful execution of an ERASE statement, currency is nullified for all record types involved in the erase both explicitly and implicitly. Run unit and area currency remain unchanged. Next, prior, and owner currencies are preserved for sets from which the last record occurrence was erased. These currencies enable you to retrieve the next or prior records within the area or the next, prior, or owner records within the set in which the erased record participated. An attempt to retrieve erased records results in an error condition.

Syntax

```
►►─── ERASE RECORD (record-name) ─┬─────────────┬─ ; ──────────────────────►◄
                                  ├─ PERMANENT ─┤
                                  ├─ SELECTIVE ─┤
                                  └─ ALL ───────┘
```

Parameters

RECORD <u>(record-name)</u>

Names the record type to be erased. *Record-name* must be a record included in the subschema. The current of *record-name* must be current of run unit. Unless the PERMANENT, SELECTIVE, or ALL qualifier follows, an error condition results if the named record is the owner of any nonempty set occurrences.

Native VSAM users: ERASE RECORD (*record-name*) is the only form of the ERASE statement valid for records in a native VSAM key-sequenced data sets (KSDS) or relative-record data sets (RRDS); the ERASE statement is not valid for a native VSAM entry-sequenced data sets (ESDS).

PERMANENT

Erases the specified record and all mandatory member record occurrences owned by the specified record. Optional member records are disconnected. If any of the erased mandatory members are themselves the owner of any set occurrences, the ERASE statement is executed on such records as if they were directly the object record of an ERASE PERMANENT statement (that is, all mandatory members of such sets are also erased). This process continues until all direct and indirect members have been processed.

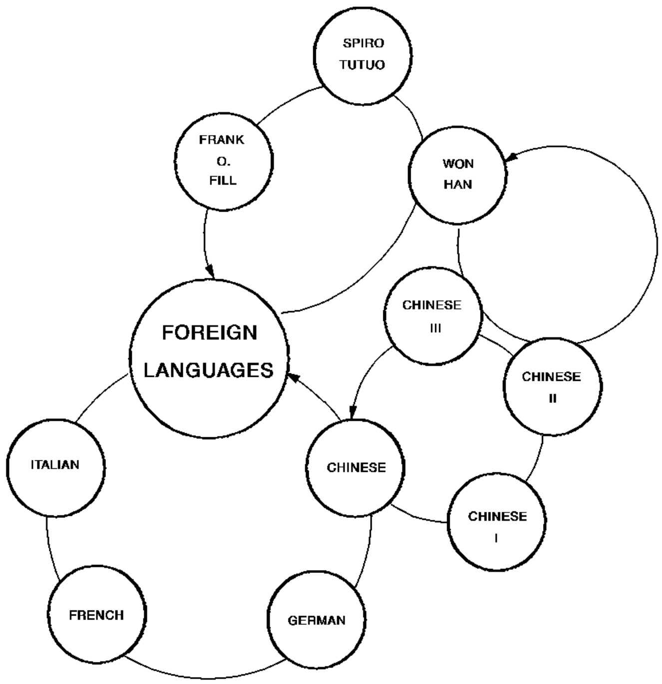
SELECTIVE

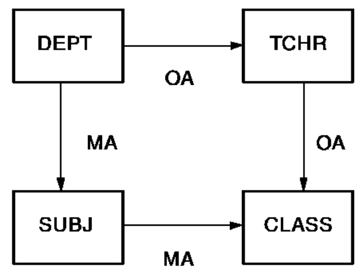
Erases the specified record and all mandatory member record occurrences owned by the specified record. Optional member records are erased if they do not *currently participate* as members in other set occurrences. All erased member records that are themselves the owners of any set occurrences are treated as if they were the object of an ERASE SELECTIVE statement.

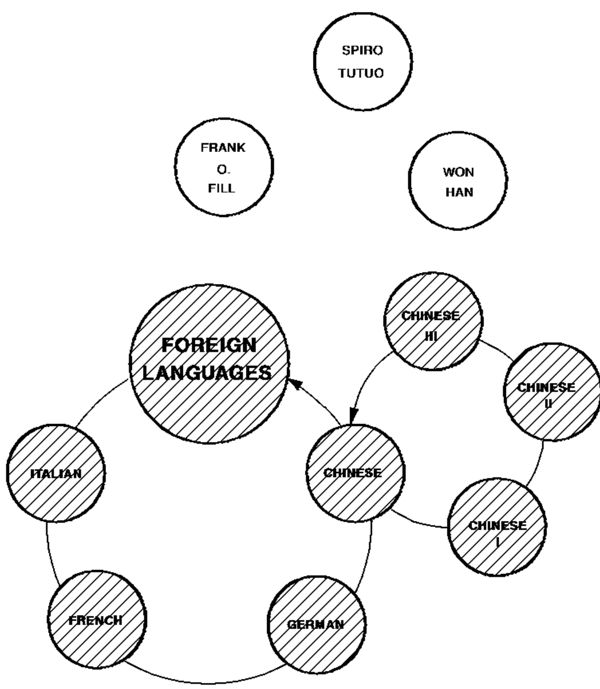
ALL

Erases the specified record and all mandatory and optional member record occurrences owned by the specified record. All erased member records that are themselves the owners of any set occurrences are treated as if they were the object record of an ERASE ALL statement.

Example: The following four figures illustrate use of the three parameters of the ERASE statement. Note that the outcome of the ERASE statement varies based on the qualifier specified (PERMANENT, SELECTIVE, or ALL). Although all three qualifiers cause all mandatory members owned by the specified record to be erased, they differ in their effect on optional members.





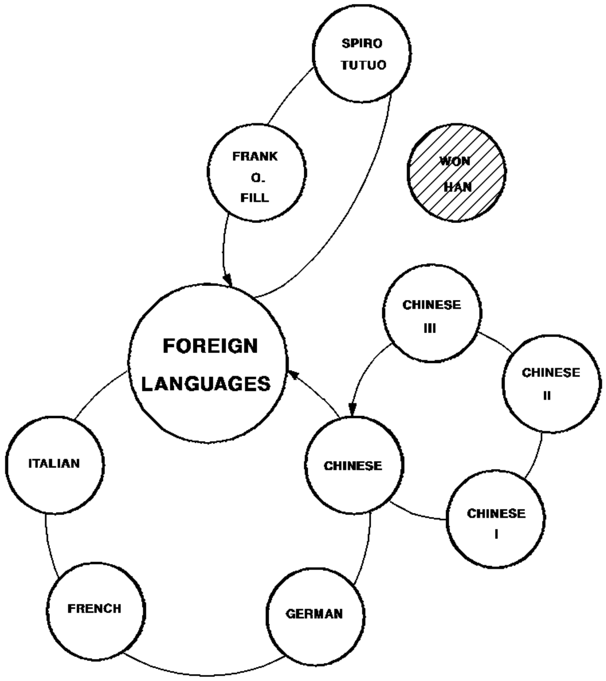


ERASE RECORD (DEPT) PERMANENT;

(assuming that FOREIGN LANGUAGES is current of run unit)

The Foreign Languages Department can no longer be funded, so it is deleted from the database along with its subjects and classes. The teachers will be reassigned to other departments.

Erases the foreign language record and all mandatory members; disconnects optional members.

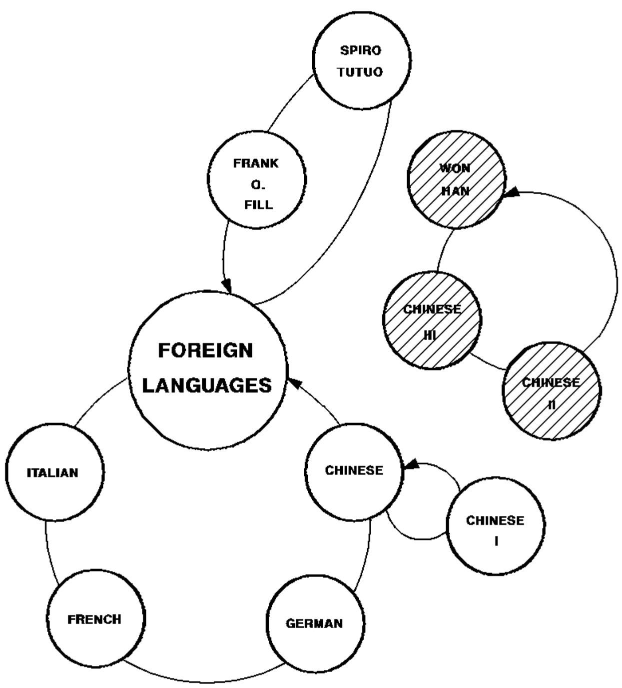


ERASE RECORD (TCHR) SELECTIVE;

(assuming that WON HAN is current of the run unit)

Won Han has quit in the middle of the semester. His classes will be finished by another teacher, so only Won Han is erased. (Remember that an unqualified ERASE command cannot be used to erase the owner of a non-empty set.)

Erases the TCHR record occurrence, mandatory members (none, TCHR_CLASS is OA), and optional orphans (none, CHI is in the SUBJ_CLASS set).



ERASE RECORD (TCHR) ALL;

(assuming that Won Han is current of run unit)

No one is available to teach Won Han's classes, so both he and his classes are deleted from the database.

Erases the TCHR record occurrence and all mandatory and optional members.

The following figure shows the effect each of the parameters has on currency.

	CURRENCIES: RUN UNIT, RECORD, SET, AREA									
	RUN UNIT	DEPT	TCHR	SUBJ	CLASS	DEPT_ TCHR	DEPT_ SUBJ	TCHR_ CLASS	SUBJ_ CLASS	SCHOOL_ REGION
ESTABLISHED CURRENCIES	FOREIGN LANG.	FOREIGN LANG.		FRENCH	CHI I.	FOREIGN LANG.	FOREIGN LANG.	CHI I.	FRENCH	FOREIGN LANG.
ERASE DEPT PERMANENT	FOREIGN LANG.	NULL		NULL	NULL	NP	NULL	NP	NULL	FOREIGN LANG.
ESTABLISHED CURRENCIES	WON HAN	FOREIGN LANG.	WON HAN		CHI I.	WON HAN	FOREIGN LANG.	WON HAN	CHI I.	WON HAN
ERASE TCHR SELECTIVE	WON HAN	FOREIGN LANG.	NULL		CHI I.	NP	FOREIGN LANG.	NP	CHI I.	WON HAN
ESTABLISHED CURRENCIES	WON HAN		WON HAN	FRENCH		WON HAN	FRENCH	WON HAN	FRENCH	WON HAN
ERASE TCHR ALL	WON HAN		NULL	FRENCH	NULL	NP	FRENCH	NP	NP	WON HAN

Status codes: Upon completion of the ERASE function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0208	The object record is not in the specified subschema.
0209	The named record's area has not been readied in one of the three update usage modes.
0210	The subschema specifies an access restriction that prohibits use of the ERASE statement.
0213	A current record of run unit has either not been established or has been nullified by a previous ERASE statement.
0217	A db-key has been encountered that contains a longterm permanent lock.
0220	The current record of run unit is not the same record type as the named record.
0221	An area other than the area of the specified record has been readied with an incorrect usage mode.
0225	Currency has not been established. Only OBTAIN statements update index set currencies.
0226	A broken chain has been encountered in the process of executing an ERASE ALL, PERMANENT, or SELECTIVE.

Status code	Meaning
0230	An attempt has been made to erase the owner record of a nonempty set.
0233	Either erasure of the record occurrence is not allowed in this subschema or all sets in which the record participates have not been included in the subschema.
0260	A record occurrence has been encountered whose type is inconsistent with the set named in the ERROR_SET field of the IDMS-DB communications block; probable causes are a broken chain or improper database description.
0261	No record can be found for a pointer db-key. The probable cause is a broken chain.

6.35 ERASE (LRF)

The ERASE statement deletes a logical-record occurrence. The ERASE statement does not necessarily result in the deletion of all or any of the database records used to create the specified logical record. The path selected to service an ERASE logical-record request performs whatever database access operations the DBA has specified to service the request. For example, if a DEPARTMENT loses an employee, the EMP_JOB_LR logical record that contains information about that employee would be erased. However, only the information about the former employee would be erased from the database, not all the information about the department; that is, EMPLOYEE information would be erased, but not DEPARTMENT, JOB, or OFFICE information.

LRF uses field values present in the variable-storage location reserved for the logical record to update the database. You can specify an alternative storage location from which LRF is to take field values to make the appropriate updates to the database.

Syntax:

```
►►── ERASE RECORD (logical-record-name) ──────────────────────────────►

►─┬──────────────────────────┬─┬────────────────────────────────┬────►
  └─ FROM (alt-logical-record) ┘ └─ WHERE (boolean-expression) ─┘

►─┬────────────────────────────────────────────────┬─ ; ──────────►◄
  └─ ON LR_STATUS (path-status)  imperative-statement ┘
```

RECORD (logical-record-name)
: Names the logical record to be deleted. Unless the FROM clause (see below) is included, LRF uses field values present in the variable-storage location reserved for the logical record to make any necessary updates to the database. *Logical-record-name* must specify a logical record defined in the subschema.

FROM (alt-logical-record)
: Names an alternative variable-storage location from which LRF is to obtain field values to perform the appropriate database updates in response to this request. When erasing a logical record that has been previously retrieved into an alternative storage location, use the FROM clause to name the same location specified in the OBTAIN request. If the FROM clause is included in the ERASE statement, *alt-logical-record* must identify a record location defined in program variable storage.

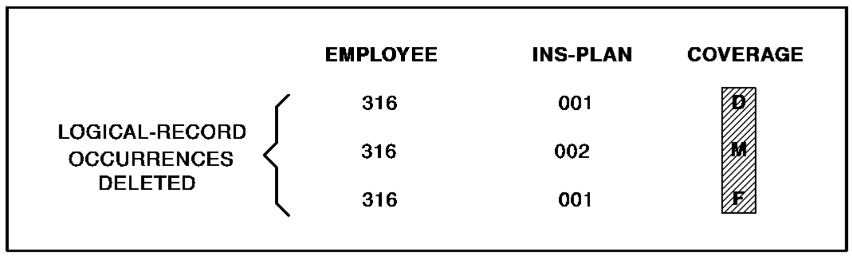
WHERE (boolean-expression)
: Specifies the selection criteria to be applied to the specified logical record. For details on coding this clause, see 6.78, “Logical-record clauses (WHERE and ON)” on page 6-215 at the end of this chapter.

ON LR_STATUS (path-status) imperative-statement
: Specifies the action to be taken if *path-status* is returned to the LR_STATUS field in the LRC block. *Path-status* must be a 1- to 16-character alphanumeric value. For details on coding this clause, see 6.78, “Logical-record clauses (WHERE and ON)” on page 6-215 at the end of this chapter.

Example: The following example illustrates a request to erase all occurrences of a former employee's EMP_INSURANCE_LR logical record. The DBA-designated path status ALL_ERASED indicates that all occurrences of the EMP_INSURANCE_LR logical record have been erased.

```
ERASE RECORD (EMP_INSURANCE_LR)
 WHERE (EMP_ID_0415 EQ '0316')
 ON LR_STATUS (ALL_ERASED) CALL EMP_INS_DELETION_RPT;
```

D, M, and F under Coverage in the following figure are physically erased from the database as a result of the ERASE RECORD (EMP_INSURANCE_LR) statement. As defined by the DBA, the ERASE EMP_INSURANCE_LR path group *logically* deletes all of the specified EMP_INSURANCE_LR occurrences, but *physically* deletes only the D, M, and F COVERAGE records.



6.36 FIND/OBTAIN

The FIND statement locates a record occurrence in the database; the OBTAIN statement locates a record and moves the data associated with the record to the record buffers. Because the FIND and OBTAIN command statements have identical formats, they are discussed together.

Six FIND/OBTAIN formats: The six formats of the FIND/OBTAIN statement are as follows:

- **FIND/OBTAIN CALC/DUPLICATE** accesses a record occurrence by using its CALC key value.
- **FIND/OBTAIN CURRENT** accesses a record occurrence by using established currencies.
- **FIND/OBTAIN DBKEY** accesses a record occurrence by using its database key.
- **FIND/OBTAIN OWNER** accesses the owner record of a set occurrence.
- **FIND/OBTAIN WITHIN SET USING SORT KEY** accesses a record occurrence in a sorted set by using its sort-key value.
- **FIND/OBTAIN WITHIN SET/AREA** accesses a record occurrence based on its logical location within a set or on its physical location within an area.

Each format of the FIND/OBTAIN statement is discussed separately in the following subsections.

SHARED and EXCLUSIVE locks: You can place locks on located record occurrences by using the KEEP clause of a FIND/OBTAIN statement. The KEEP clause sets a shared or exclusive lock:

- **KEEP** places a shared lock on the located record occurrence. Other concurrently executing run units can access but not update the locked record.
- **KEEP EXCLUSIVE** places an exclusive lock on the located record occurrence. Other concurrently executing run units can neither access nor update the locked record.

For more information on record locks, see 6.46, “KEEP” on page 6-116, later in this chapter.

6.36.1 FIND/OBTAIN CALC/DUPLICATE

The FIND/OBTAIN CALC/DUPLICATE statement locates a record based on the value of an element defined as a CALC key in the record. The specified record must be stored in the database with a location mode of CALC. Before issuing the FIND/OBTAIN CALC/DUPLICATE statement, you must initialize a field in program variable storage with the CALC-key value.

You can use the DUPLICATE option to access duplicate records with the same CALC-key value as the record that is current of record type, provided that a FIND/OBTAIN CALC statement has previously accessed an occurrence of the same record type.

Currency: Following successful execution of a FIND/OBTAIN CALC/DUPLICATE statement, the accessed record becomes the current record of run unit, its record type, its area, and all sets in which it currently participates as member or owner.

Syntax:

```
►►──┬─ FIND ───┬──┬──────────────────────────┬──┬─┬─ CALC ─┬─┬──────────►
    └─ OBTAIN ─┘  └─ KEEP ─┬───────────────┬─┘  │ └─ ANY ──┘ │
                           └─ EXCLUSIVE ───┘    └─ DUPLICATE ─┘

 ►─── RECORD (record-name); ───────────────────────────────────────────►◄
```

Parameters

FIND/OBTAIN CALC/DUPLICATE RECORD (record-name)
Locates the record specified by *record-name* based on its CALC-key value:

CALC/ANY
Locates the first or only occurrence of the designated record type whose CALC key matches the value of the CALC data item in program variable storage. CALC and ANY are synonyms.

DUPLICATE
Locates the next record with the same CALC key value as the current of record type. Use of the DUPLICATE option requires prior selection of an occurrence of the same record type with the CALC option. If the value of the CALC key in variable storage is not equal to the CALC-key field of the current of record type, an error status of 0332 is returned.

KEEP EXCLUSIVE
Places a shared (KEEP) or exclusive (KEEP EXCLUSIVE) lock on the accessed record.

Example: To retrieve an occurrence of the EMPLOYEE record by using the FIND/OBTAIN CALC/DUPLICATE statement, you must first initialize the variable-storage field that contains the CALC-control element. The following statements initialize the CALC field EMP_ID_0415 and retrieve an occurrence of the EMPLOYEE record:

```
EMP_ID_0415 = EMP_ID_IN;
OBTAIN CALC RECORD (EMPLOYEE);
```

Status codes: Upon completion of the FIND/OBTAIN CALC/DUPLICATE function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0301	The area in which the named record participates has not been readied.
0306	A successful FIND/OBTAIN CALC has not yet been executed (applies to the DUPLICATE option only).
0308	The named record is not in the subschema. The program probably invoked the wrong subschema.
0310	The subschema specifies an access restriction that prohibits retrieval of the named record.
0318	The record has not been bound.
0326	The record cannot be found or no more duplicates exist for the named record.
0331	The retrieval statement format conflicts with the record's location mode.
0332	The value of the CALC data item in program variable storage does not equal the value of the CALC data item in the current record (applies to the DUPLICATE option only).
0364	The CALC-control element has not been described correctly either in the program or in the subschema.
0370	A database file will not open properly.

If the KEEP parameter is specified in a FIND/OBTAIN statement, and an error occurs during KEEP processing, the major code 06 is returned. For further information, see 6.46, “KEEP” on page 6-116 later in this chapter. The major code 03 is returned if an error occurs during FIND/OBTAIN processing.

6.36.2 FIND/OBTAIN CURRENT

The FIND/OBTAIN CURRENT statement locates the record that is current of its record type, set, or area. This form of the FIND/OBTAIN statement is an efficient means of establishing the appropriate record as current of run unit before executing a DML statement that utilizes run-unit currency (for example, ACCEPT, IF, GET, MODIFY, ERASE).

Currency: Following successful execution of a FIND/OBTAIN CURRENT statement, the accessed record is current of run unit, its record type, its area, and all sets in which it currently participates as member or owner.

Syntax:

```
►►─┬─ FIND ───┬─┬──────────────────────┬─ CURRENT ────────────────►
   └─ OBTAIN ─┘ └─ KEEP ─┬───────────┬─┘
                         └─ EXCLUSIVE ─┘

►─┬──────────────────────┬─ ; ─────────────────────────────────►◄
  ├─ RECORD (record-name) ─┤
  ├─ SET (set-name) ───────┤
  └─ AREA (area-name) ─────┘
```

Parameters

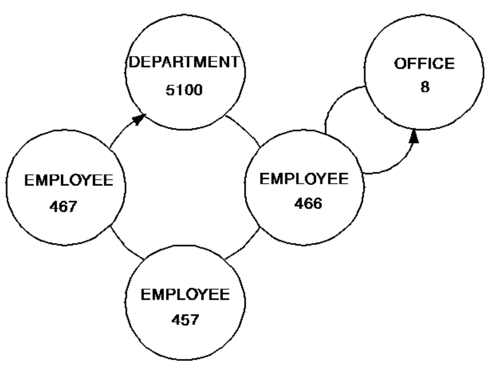
FIND/OBTAIN CURRENT
Locates the current record occurrence of a specified record type, set, or area.

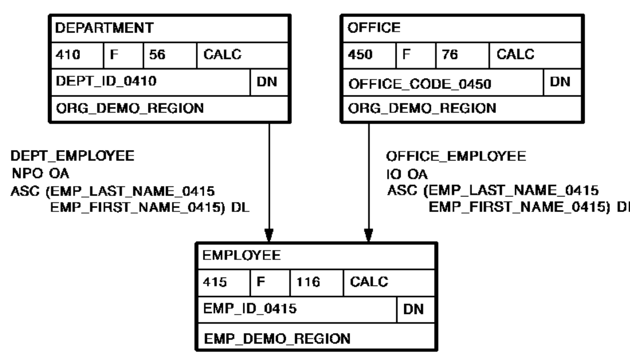
KEEP EXCLUSIVE
Places a shared (KEEP) or exclusive (KEEP EXCLUSIVE) lock on the accessed record.

RECORD (record-name)/SET *(set-name)***/AREA** *(area-name)*
Specifies that the current record of the named record type, set, or area is to be accessed.

Example: The following figure illustrates use of the FIND/OBTAIN CURRENT statement to establish the proper record as current of run unit before the record is modified.

Assume that you enter the database on DEPARTMENT 5100 by using CALC retrieval. You examine EMPLOYEE 466 by using within set retrieval and obtain further information from its owner OFFICE record (OFFICE 8). OFFICE 8 becomes current of run unit. Before modifying EMPLOYEE 466, you must issue the FIND CURRENT statement to reestablish EMPLOYEE 466 as current of run unit.





	CURRENCIES RUN UNIT, RECORD, SET, AREA							
	RUN UNIT	DEPARTMENT	EMPLOYEE	OFFICE	DEPT_EMPLOYEE	OFFICE_EMPLOYEE	ORG_DEMO_REGION	EMP_DEMO_REGION
DEPT_ID=5100 FIND CALC RECORD (DEPARTMENT);	5100	5100			5100		5100	
OBTAIN FIRST SET (DEPT_EMPLOYEE);	466	5100	466		466	466	5100	466
OBTAIN OWNER SET (OFFICE_EMPLOYEE);	8	5100	466	8	466	8	8	466
FIND CURRENT RECORD (EMPLOYEE);	466	5100	466	8	466	466	8	466
MODIFY RECORD (EMPLOYEE);	466	5100	466	8	466	466	8	466

►► For a complete description of the MODIFY statement and its use, see 6.53, "MODIFY RECORD" on page 6-146, later in this chapter.

Status codes: Upon completion of the FIND/OBTAIN CURRENT function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0301	The area in which the named record participates has not been readied.
0306	Currency has not been established for the named record, set, or area.
0308	The named record or set is not in the subschema. The program has probably invoked the wrong subschema.
0310	The subschema specifies an access restriction that prohibits retrieval of the named record.
0313	A current record of run unit either has not been established or has been nullified by a previous ERASE statement.
0323	The specified area name has not been included in the subschema invoked.

If the KEEP parameter is specified in a FIND/OBTAIN statement, and an error occurs during KEEP processing, the major code 06 is returned. For further information, see 6.46, “KEEP” on page 6-116, later in this chapter. The major code 03 is returned if an error occurs during FIND/OBTAIN processing.

6.36.3 FIND/OBTAIN DBKEY

The FIND/OBTAIN DBKEY statement locates a record occurrence directly by using a database key that has been stored previously by the program. The DML ACCEPT statement, discussed earlier in this chapter, or the PL/I assignment statement can be used to save a db-key. Any record in the program's subschema can be accessed directly in this manner, regardless of its location mode.

Native VSAM users: This statement is not valid for accessing data records in a native VSAM key-sequenced data set (KSDS).

Currency: After successful execution of a FIND/OBTAIN DBKEY statement, the accessed record becomes the current record of run unit, its record type, its area, and all sets in which it currently participates as member or owner. In addition, the RECORD_NAME field of the IDMS-DB communications block is updated with the name of the accessed record.

Note that currency is not used to determine the specified record of the FIND/OBTAIN DBKEY statement; the record is identified by its db-key and, optionally, by its record type.

Syntax

```
►►─┬─ FIND ───┬─┬──────────────────────────┬──────────────────────────────►
   └─ OBTAIN ─┘ └─ KEEP ─┬───────────────┬─┘
                         └─ EXCLUSIVE ─┘

►─┬─ DBKEY (db-key-v) ─┬──────────────────────────┬─┬─────────────────────►◄
  │                    └─ PAGE_INFO (page-info-v) ─┘ │
  └─┬──────────────────────────┬─ DBKEY (db-key-v) ──┘
    └─ RECORD (record-name) ─┘
```

Parameters

FIND/OBTAIN DBKEY (db-key-v)

Locates a record directly by using a db-key value contained in program variable storage. *(db-key-v)* is a FIXED BINARY(31) fullword field that identifies the location in program variable storage that contains a db-key previously saved by the program.

If a record name has been specified, *(db-key-v)* must contain the db-key of an occurrence of the named record type.

If a record name has not been specified and the subschema includes areas with different page information values, then:

- If PAGE_INFO has been specified, *(db-key-v)* must contain the db-key of an occurrence of a record type whose page information matches that specified.

- If PAGE_INFO has not been specified, *(db-key-v)* must contain the db-key of an occurrence of a record type whose page information matches that of the record that is current of run unit.

If a record name has not been specified and all areas in the subschema have the same page information value, *(db-key-v)* can contain the db-key of an occurrence of any record type in the subschema.

KEEP EXCLUSIVE
Places a shared (KEEP) or exclusive (KEEP EXCLUSIVE) lock on the accessed record.

PAGE_INFO (page-info-v)
Specifies page information that is used to determine the area with which the db-key is associated. If neither record name nor PAGE_INFO is specified and the subschema includes areas with different page information values, the page information associated with the record that is current of rununit is used.

Note: Page information is only used if the subschema includes areas with different page information values; otherwise, it is ignored.

page-info-v is a field that identifies the location within program variable storage containing the page information associated with the specified db-key. It may be defined either as a fullword field or as a group field consisting of two halfwords.

RECORD (record-name)
Optionally identifies the record type of the requested record. If specified, *record-name* must name a record that is included in the subschema.

Example: The following statement locates the occurrence of the HOSPITAL_CLAIM record whose db-key matches the value of a field in program variable storage called SAVED_KEY:

```
FIND RECORD (HOSPITAL_CLAIM) DBKEY (SAVED_KEY);
```

The located record becomes current of run unit, current of the HOSPITAL_CLAIM record type, current of the INS_DEMO_REGION area, and current of the COVERAGE_CLAIMS set.

Status codes: Upon completion of the FIND/OBTAIN DBKEY function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0301	The area in which the named record participates has not been readied.
0302	The db-key is inconsistent with the area in which the record is stored. Either the db-key has not been initialized properly or the record name is incorrect.

Status code	Meaning
0308	The named record is not in the subschema. The program has probably invoked the wrong subschema.
0310	The subschema specifies an access restriction that prohibits retrieval of the named record.
0326	The record cannot be found; record occurrence not correct type
0370	A database file will not open properly.
0371	The requested page cannot be found in the DMCL.

If the KEEP parameter is specified in a FIND/OBTAIN statement, and an error occurs during KEEP processing, the major code 06 is returned. For further information, see 6.46, “KEEP” on page 6-116, later in this chapter. The major code 03 is returned if an error occurs during FIND/OBTAIN processing.

6.36.4 FIND/OBTAIN OWNER

The FIND/OBTAIN OWNER statement locates the owner record of the current occurrence of a set. This statement can be used to retrieve the owner record of any set whether or not that set has been assigned owner pointers.

Native VSAM users: The FIND/OBTAIN OWNER statement is not valid since owner records are not defined in native VSAM data sets.

Currency: In order to execute a FIND/OBTAIN OWNER statement, currency must be established for the specified set.

Note: When a record declared as an optional or manual member of a set is retrieved, it is *not* established as current of set if it is not currently connected to the specified set. A subsequent attempt to retrieve the owner record will locate instead the owner of the current record of set. In such cases, you should determine whether the retrieved record is actually a member in the specified set before executing the FIND/OBTAIN OWNER statement. The IF MEMBER statement, explained later in this chapter, can be used for this purpose.

Following successful execution of a FIND/OBTAIN OWNER statement, the accessed record becomes the current record of run unit, its record type, its area, and all sets in which it currently participates as member or owner. If the current record of set is the owner record when the statement is executed, currency within the specified set remains unchanged.

Syntax:

```
►►─┬─ FIND ───┬─┬──────────────────────────┬─ OWNER SET (set-name); ──────────►◄
   └─ OBTAIN ─┘ └─ KEEP ─┬─────────────┬─┘
                         └─ EXCLUSIVE ─┘
```

Parameters

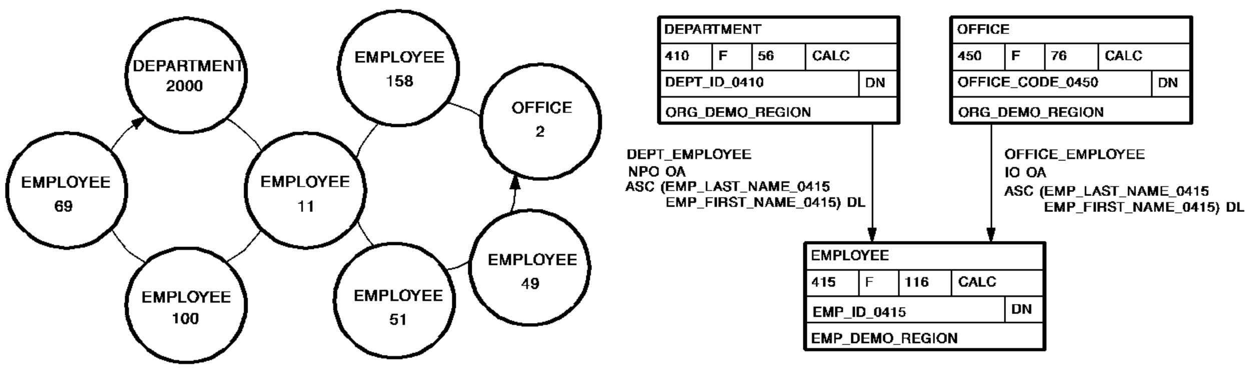
FIND/OBTAIN OWNER SET (set-name)

Specifies the set whose owner record is to be retrieved. *Set-name* must be a set included in the subschema.

KEEP EXCLUSIVE

Places a shared (KEEP) or exclusive (KEEP EXCLUSIVE) lock on the accessed record.

Example: The following figure illustrates use of the FIND/OBTAIN OWNER statement to move through the database.



CURRENCIES
RUN UNIT, RECORD, SET, AREA

	RUN UNIT	DEPARTMENT	EMPLOYEE	OFFICE	DEPT_EMPLOYEE	OFFICE_EMPLOYEE	ORG_DEMO_REGION	EMP_DEMO_REGION
DEPT_ID = 2000 OBTAIN CALC RECORD (DEPARTMENT);	2000	2000			2000		2000	
OBTAIN FIRST SET (DEPT_EMPLOYEE);	11	2000	11		11	11	2000	11
OBTAIN OWNER SET (OFFICE_EMPLOYEE);	2	2000	11	2	11	2	2	11

Status codes: Upon completion of the FIND/OBTAIN OWNER function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.

Status code	Meaning
0301	The area in which the object record participates has not been readied.
0306	Currency has not been established for the record, set, or area.
0308	The named set is not in the subschema. The program has probably invoked the wrong subschema.
0310	The subschema specifies an access restriction that prohibits retrieval of the object record.
0360	A record occurrence has been encountered whose record type is not a member or owner of the set as it is defined in the subschema.
0370	A database file will not open properly.

If the KEEP parameter is specified in a FIND/OBTAIN statement, and an error occurs during KEEP processing, the major code 06 is returned. For further information, see 6.46, “KEEP” on page 6-116, later in this chapter. The major code 03 is returned if an error occurs during FIND/OBTAIN processing.

6.36.5 FIND/OBTAIN WITHIN SET USING SORT KEY

The FIND/OBTAIN WITHIN SET USING SORT KEY statement locates a member record in a sorted set. Sorted sets are ordered in ascending or descending sequence based on the value of a sort-control element in each member record. The search begins with either the current of set or the owner of the current of set and always proceeds through the set in the next direction.

Before issuing this statement, you must initialize the sort-control element in program variable storage. The record occurrence selected will have a key value equal to the value of the sort-control element. If more than one record occurrence contains a sort key equal to the key value in variable storage, the first such record will be selected.

You can use FIND/OBTAIN WITHIN SET USING SORT KEY to access both sorted chained sets and sorted index sets.

Note: In a batch environment, sorted sets can be processed more efficiently by sorting the input transactions.

Currency: Following successful execution of a FIND/OBTAIN WITHIN SET USING SORT KEY statement, the accessed record becomes current of run unit, its record type, its area, and all sets in which it currently participates as member or owner. If a member record with the requested sort-key value is not found, the current of set is nullified but the next of set and prior of set are maintained. The next of set is the member record with the next higher sort-key value (or next lower for descending sets) than the requested value; the prior of set is the member record with the next lower value (or higher for descending sets) than requested. Because these currencies are maintained, the program can walk the set to do a generic search on the sort-key value.

Syntax:

```
►►─┬─ FIND ───┬─┬──────────────────────────┬─ RECORD (record-name) ─────────►
   └─ OBTAIN ─┘ └─ KEEP ─┬───────────────┬─┘
                         └─ EXCLUSIVE ───┘

 ►── SET (set-name) ─┬─────────────┬─ USING (sort-field-name); ───────────►◄
                     └─ CURRENT ───┘
```

Parameters

FIND/OBTAIN RECORD (record-name) SET (set-name)
Specifies the record type and sorted set name. The search begins with the *owner* of the current record of the specified set.

KEEP EXCLUSIVE
Places a shared (KEEP) or exclusive (KEEP EXCLUSIVE) lock on the accessed record.

CURRENT
Indicates that the search begins with the currencies already established for the specified set.

If the key value for the record that is current of set is higher than the key value of the requested record (assuming ascending set order), a NOT FOUND condition results. In a descending set order, if the key value for the record that is current of set is lower than the key value of the requested record, a NOT FOUND condition results.

USING (sort-field-name)
Specifies the sort-control element to be used in searching the sorted set. *Sort-field-name* is either the name of the sort-control element in the record or the symbolic name of a field in variable storage that contains the value of the sort-control element.

Example: The following example illustrates the use of a FIND/OBTAIN WITHIN SET USING SORT KEY statement. Assume that the SKILL_NAME_NDX set is ordered in ascending sequence based on the value stored in SKILL_NAME_0455 in each SKILL record occurrence. Retrieval of a SKILL record with a skill name equal to PL/I is accomplished by coding the following statements:

```
SKILL_NAME_0455 = 'PL/I';
FIND RECORD (SKILL) SET (SKILL_NAME_NDX)
                   USING (SKILL_NAME_0455);
```

Status codes: Upon completion of the FIND/OBTAIN WITHIN SET USING SORT KEY function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0057	A retrieval-only run unit has detected an inconsistency in an index that should cause an 1143 abend, but optional APAR bit 216 has been turned on.
0301	The area in which the named record participates has not been readied.
0306	Currency has not been established for the named set.
0308	Either the named record or set is not in the subschema or the named record is not a member of the named set. The program has probably invoked the wrong subschema.
0310	The subschema specifies an access restriction that prohibits retrieval of the named record.
0326	The record cannot be found.
0331	The retrieval statement format conflicts with the record's location mode.
0360	A record occurrence has been encountered whose record type is not a member or owner of the set as it is defined in the subschema.
0370	A database file will not open properly.

If the KEEP parameter is specified in a FIND/OBTAIN statement, and an error occurs during KEEP processing, the major code 06 is returned. For further information, see 6.46, “KEEP” on page 6-116, later in this chapter. The major code 03 is returned if an error occurs during FIND/OBTAIN processing.

6.36.6 FIND/OBTAIN WITHIN SET/AREA

The FIND/OBTAIN WITHIN SET/AREA statement locates records either logically, based on set relationships, or physically, based on database location. The formats of this statement allow you either to access serially each record in a set or area or to select specific occurrences of a given record type within the set or area.

Selecting from a set: The following rules apply to the selection of member records within a *set*:

- The set occurrence used as the basis for the operation is determined by the current record of the specified set. Set currency must be established before attempting to access records within a set.
- The next or prior record within a set is the subsequent or previous record relative to the *current record of the named set* in the logical order of the set. The prior record in a set can be retrieved only if the set has been assigned prior pointers.

- The first or last record within a set is the first or last member occurrence in terms of the logical order of the set. The selected record is the same as would be selected if the current of set were the owner record and the next or prior record had been requested. The last record in a set can be retrieved only if the set has prior pointers.
- The *n*th occurrence of a record within a set can be retrieved by specifying a sequence number that identifies the position of the record in the set. The DBMS begins its search with the *owner of the current of set* for the specified set and continues until it locates the *n*th record or encounters an end-of-set condition. If the specified sequence number is negative, the search proceeds in the prior direction within the set. A negative sequence number can be used only if the set has prior pointers; a sequence number of 0 produces an error status of 0304.
- When an end-of-set condition occurs, the owner record occurrence of the set becomes the current record of run unit, current of its record type, current of its area, and current record of *only* the set involved in this operation. Currency of other sets in which the specified record participates as owner or member remains unaffected.

  Note: If OBTAIN has been specified, the contents of the owner record are not moved to program variable storage (that is, OBTAIN under these circumstances is treated as a FIND).

  Native VSAM users: When an end-of-set condition occurs, all currencies remain unchanged.

Selecting from an area: The following rules apply to the selection of records within an *area*:

- The first record occurrence within an area is the one with the lowest database key; the last record is the one with the highest database key.
- The next record within an area is the one with the next higher database key relative to the *current record of the named area*; the prior record is the one with the next lower database key relative to the current of area.
- The first or last or *n*th record in an area must be retrieved to establish the correct starting position before next or prior records are requested.

Currency: Following successful execution of a FIND/OBTAIN WITHIN SET/AREA statement, the accessed record becomes the current record of run unit, its record type, its area, and all sets in which it currently participates as member or owner.

When an end-of-set condition occurs selecting records within a set, the owner record occurrence of the set becomes the current record of run unit, its record type, its area, and *only* the set involved in this operation. Currency of other sets in which the specified record participates as owner or member remains unaffected.

Syntax:

```
►►─┬─ FIND ───┬──┬────────────────────────────┬─┬─ NEXT ──────────────────┬──────────►
   └─ OBTAIN ─┘  └─ KEEP ─┬─────────────────┬─┘ ├─ FIRST ─────────────────┤
                          └─ EXCLUSIVE ─┘       ├─ LAST ──────────────────┤
                                                ├─ PRIOR ─────────────────┤
                                                └─ NTH (sequence-number) ─┘

►─┬──────────────────────────┬─┬─ SET (set-name) ───┬─ ; ──────────────────────►◄
  └─ RECORD (record-name) ───┘ └─ AREA (area-name) ─┘
```

Parameters

FIND/OBTAIN SET (set-name)/AREA (area-name)
Locates a record based on its location within a set or area. *Set-name/area-name* specifies the set or area that will be searched and must identify a set or area included in the subschema.

KEEP EXCLUSIVE
Places a shared (KEEP) or exclusive (KEEP EXCLUSIVE) lock on the accessed record.

NEXT
Accesses the next record in the specified set or area relative to the current record.

FIRST
Accesses the first record in the specified set or area.

LAST
Accesses the last record in the specified set or area. The specified set must have prior pointers.

PRIOR
Accesses the prior record in the specified set or area relative to the current record. The specified set must have prior pointers.

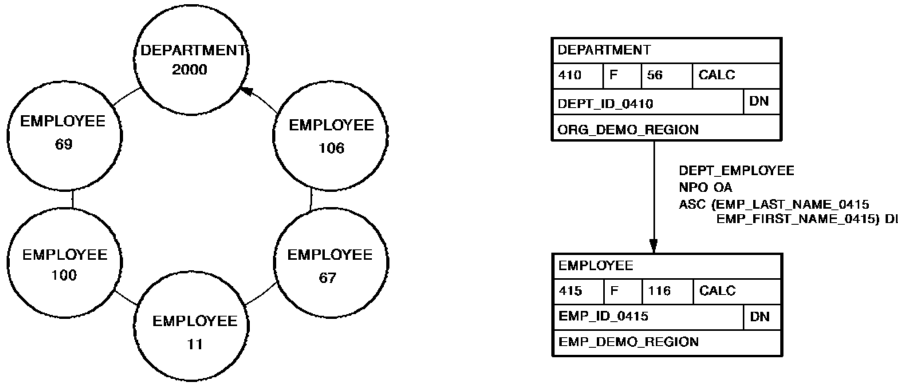
NTH (sequence-number)
Accesses the *n*th record in the specified set or area. *Sequence-number* must either be a positive or negative number or any numeric field that contains a nonzero value used by the DBMS in searching for the *n*th record occurrence. If *sequence* is negative, the specified set must have prior pointers.

Native VSAM users: FIRST, LAST, and NTH *(sequence)* options are not valid for a native VSAM KSDS with spanned records.

RECORD (record-name)
Specifies that within a set or area, only occurrences of the named record type will be accessed. *Record-name* must be defined as a member of the specified set or contained within the specified area.

Example: The following figure illustrates the retrieval of records in an occurrence of the DEPT_EMPLOYEE set.

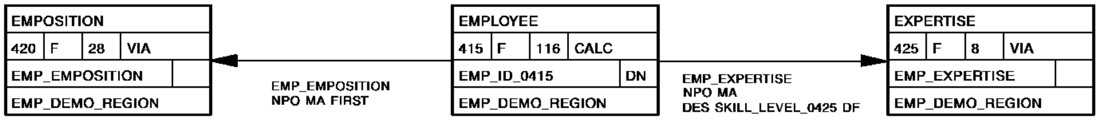
The FIND CALC statement establishes currency in the DEPT_EMPLOYEE set. Member EMPLOYEE records are then retrieved by a series of OBTAIN WITHIN SET statements. EMPLOYEE 106 is the last record in the set and the next OBTAIN statement returns an end-of-set condition, positioning run-unit currency at the owner of the set, DEPARTMENT 2000.



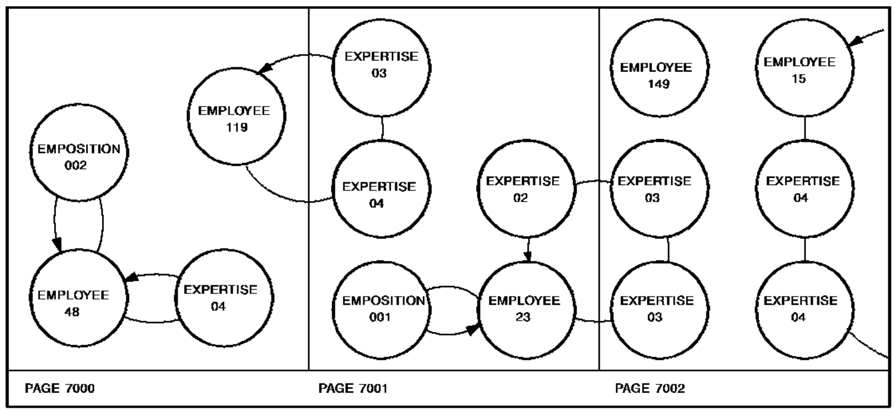
	CURRENCIES RUN UNIT, RECORD, SET, AREA							
	RUN UNIT	DEPARTMENT	EMPLOYEE	DEPT_EMPLOYEE	OFFICE_EMPLOYEE	ORG_DEMO_REGION	EMP_DEMO_REGION	
DEPT_ID = 2000 FIND CALC RECORD (DEPARTMENT);	2000	2000		2000		2000		
OBTAIN FIRST SET (DEPT_EMPLOYEE);	69	2000	69	69	69	2000	69	
OBTAIN NEXT SET (DEPT_EMPLOYEE);	100	2000	100	100	100	2000	100	
OBTAIN NTH (5) SET (DEPT_EMPLOYEE);	106	2000	106	106	106	2000	106	
OBTAIN NEXT SET (DEPT_EMPLOYEE);	2000	2000	106	2000	106	2000	106	ERROR-STATUS OF '0307'

The following figure illustrates special considerations relating to the retrieval of records in an area that contains multiple record types.

A sweep of the EMP_DEMO_REGION is performed, retrieving sequentially each EMPLOYEE record and all records in the associated EMPLOYEE_EXPERTISE set. The first command retrieves EMPLOYEE 119. Subsequent OBTAIN WITHIN SET statements retrieve the associated EXPERTISE records and establish currency on EXPERTISE 03. The FIND CURRENT statement is used to reestablish the proper position before retrieving EMPLOYEE 48. If FIND CURRENT EMPLOYEE is not specified, an attempt to retrieve the next EMPLOYEE record in the area would return EMPLOYEE 23.



EMP-DEMO-REGION AREA



	CURRENCIES RUN UNIT, RECORD, SET, AREA				
	RUN UNIT	EMPLOYEE	EXPERTISE	EMP_EXPERTISE	EMP_DEMO_REGION
OBTAIN FIRST RECORD (EMPLOYEE) AREA (EMP_DEMO_REGION);	119	119		119	119
OBTAIN FIRST RECORD (EXPERTISE) SET (EMP_EXPERTISE);	04	119	04	04	04
OBTAIN NEXT RECORD (EXPERTISE) SET (EMP_EXPERTISE);	03	119	03	03	03
FIND CURRENT RECORD (EMPLOYEE);	119	119	03	119	119
OBTAIN NEXT RECORD (EMPLOYEE) AREA EMP_DEMO_REGION);	48	48	03	48	48

Status codes: Upon completion of the FIND/OBTAIN WITHIN SET/AREA function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.

Status code	Meaning
0057	A retrieval-only run unit has detected an inconsistency in an index that should cause an 1143 abend, but optional APAR bit 216 has been turned on.
0301	The area in which the named record participates has not been readied.
0304	Either a sequence number of 0 or a variable field that contains a value of 0 was specified for the named record.
0306	Currency has not been established for the named record, set, or area.
0307	Either the end of the set or the area was reached or the set is empty.
0308	Either the named record or set is not in the subschema or the named record is not defined as a member of the named set. The program has probably invoked the wrong subschema.
0310	The subschema specifies an access restriction that prohibits retrieval of the named record.
0323	Either the area name specified has not been included in the subschema invoked or the record name specified has not been defined within the named area.
0326	The record cannot be found.
0360	A record occurrence has been encountered whose record type is not a member or owner of the set as it is defined in the subschema.
0370	A database file will not open properly.

If the KEEP parameter is specified in a FIND/OBTAIN statement, and an error occurs during KEEP processing, the major code 06 is returned. For further information, see 6.46, “KEEP” on page 6-116, later in this chapter. The major code 03 is returned if an error occurs during FIND/OBTAIN processing.

6.37 FINISH

The FINISH statement relinquishes control over all database areas associated with a program or task and optionally establishes an end-of-task checkpoint for scratch and queue areas associated with a task. FINISH writes statistical information for the database operations performed during run unit execution to the journal file; it also defines and logs the end checkpoint for a recovery unit.

You can use the FINISH statement to change area usage modes defined by previously issued READY statements.

You can use the FINISH statement in both navigational and Logical Record Facility (LRF) environments.

Currency: Following the successful execution of a FINISH request, all currencies are set to null; the issuing program or task cannot perform database access without executing another BIND/READY sequence.

Syntax

```
►►── FINISH ─┬──────────┬─ ; ──────────────────────────────────────►◄
             └─ TASK ───┘
```

Parameters

FINISH
Releases all data areas held by the issuing run unit and writes an end-of-job checkpoint and statistical information to the journal file. No further DML retrieval or modification statements can be executed until the appropriate binds have been issued and the necessary areas have been readied again.

TASK
Releases all data areas held by all run units executing under the issuing task. FINISH TASK is CA-IDMS/DC only.

Example: The following statement illustrates the use of the FINISH statement:

```
FINISH;
```

Status codes: Upon completion of the FINISH function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
5031	The specified request is invalid; the program may contain a logic error.
5096	There are too many run units currently active; check the log for details.

Status code	Meaning
5097	An invalid status has been received from DBIO/DBMS; check the log for details.

6.38 FREE STORAGE (DC/UCF)

The FREE STORAGE statement instructs the system to release all or a part of a variable-storage area. The storage to be released must have been acquired by means of a GET STORAGE request in the issuing task or by another task running on the same terminal as the issuing task. A partial release is valid only for user storage; shared storage must be freed in its entirety.

Syntax

```
►►── FREE STORAGE ──────────────────────────────────────────────────►

►─┬─ STGID (storage-id) ─────────────────────────────────────┬─ ; ─►◄
  └─ FOR (storage-location) ─┬──────────────────────────────┬┘
                             └─ FROM (start-free-storage-location) ─┘
```

Parameters

STGID (storage-id)
Specifies the 4-character identifier of the variable storage area to be released. *Storage-id* is either the symbolic name of a user-defined field that contains the ID or the ID itself enclosed in single quotation marks.

FOR (storage-location)
Specifies the variable-storage entry of the storage area to be released.

FROM (start-free-storage-location)
Releases a portion of the variable-storage area defined as user storage. *Start-free-storage-location* is the symbolic name of a user-defined field that contains the starting point of the storage area to be released. The system releases storage from the specified location to the end of the storage area.

Example: The following example releases the storage area identified as 09PA:

```
FREE STORAGE STGID ('09PA');
```

Status codes: Upon completion of the FREE STORAGE function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
3213	The requested storage ID cannot be found.
3232	The derived length of the variable-storage area is zero or negative.

6.39 GET

The GET statement transfers the contents of a specified record occurrence from the record buffer into program variable storage. Elements in the specified record are moved to their respective locations in variable storage according to the subschema view of the record. The transferred elements will appear in storage at the location to which the record has been bound (for further details, see 6.16, "BIND RECORD" on page 6-34 earlier in this chapter).

Currency: The GET statement operates only on the record that is current of run unit. Following successful execution of a GET statement, the accessed record is current of run unit, its record type, its area, and all sets in which it participates as member or owner.

Syntax:

```
►►── GET ─┬─────────────────────────┬─ ; ────────────────────────────►◄
          └─ RECORD (record-name) ──┘
```

Parameter

RECORD (record-name)
Optionally specifies the record type of the current of run unit. If this optional clause is used, the current of run unit must be an occurrence of the named record type.

Example: The following statement moves the record that is current of run unit (in this case, the OFFICE record) from the record buffer into program variable storage:

```
GET RECORD (OFFICE);
```

Status codes: Upon completion of the GET function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0506	Currency has not been established.
0508	The named record is not in the subschema. The program has probably invoked the wrong subschema.
0510	The subschema specifies an access restriction that prohibits retrieval of the named record.
0513	A current record of run unit either has not been established or has been nullified by a previous ERASE statement.
0518	The record has not been bound.
0520	The current record is not the same type as the named record.

Status code	Meaning
0526	The requested record has been erased.
0555	An invalid length has been returned for a variable-length record.

6.40 GET QUEUE (DC/UCF)

The GET QUEUE statement retrieves a queue record and places it in a storage area associated with the issuing program. If the queue record is larger than the designated storage area, the record is truncated. The system automatically deletes the retrieved record from the queue unless the GET QUEUE statement explicitly keeps the record in the queue.

Syntax

```
►►─── GET QUEUE ─┬──────────────────┬─┬─ NEXT ← ──────────────────────┬──────►
                 └─ ID (queue-id) ──┘ ├─ FIRST ───────────────────────┤
                                      ├─ LAST ────────────────────────┤
                                      ├─ PRIOR ───────────────────────┤
                                      ├─ SEQUENCE (sequence-number) ──┤
                                      └─ RECORD ID (queue-record-id) ─┘

►─┬──────────────┬─┬─────────────┬─┬─────────────┬──────────────────────────►
  ├─ DELETE ← ───┤ ├─ LOCK ← ────┤ ├─ WAIT ──────┤
  └─ KEEP ───────┘ └─ NOLOCK ────┘ └─ NOWAIT ← ──┘

►─── INTO (return-queue-data-location) ──────────────────────────────────────►

►─┬─ TO (end-queue-data-location) ──────────┬────────────────────────────────►
  └─ MAX LENGTH (queue-data-max-length) ────┘

►─┬─────────────────────────────────────────────────────┬─ ; ────────────►◄
  └─ RETURN LENGTH INTO (queue-data-actual-length) ─────┘
```

Parameters

ID (queue-id)
Specifies the 1- to 16-character ID of the queue associated with the record to be retrieved. *Queue-id* is either the symbolic name of a user-defined field that contains the ID, or the ID itself enclosed in single quotation marks. If the queue ID is not specified, a null ID of 16 blanks is assumed.

NEXT/FIRST/LAST/PRIOR/SEQUENCE *(sequence)***/RECORD ID***(queue-record-id)*
Specifies the queue record to be retrieved:

NEXT
Retrieves the next record in the queue. If currency has not been established, NEXT is equivalent to FIRST. NEXT is the default.

FIRST
Retrieves the first record in the queue.

LAST
Retrieves the last record in the queue.

PRIOR
Retrieves the prior record in the queue. If currency has not been established, PRIOR is equivalent to LAST.

SEQUENCE <u>(sequence)</u>
Retrieves the queue record identified by *sequence*. *Sequence* is either the symbolic name of a user-defined field that contains the sequence number of the record, or the sequence number itself expressed as a numeric constant.

RECORD ID <u>(queue-record-id)</u>
Retrieves the record identified by *queue-record-id*. *Queue-record-id* is the symbolic name of the FIXED BINARY(31) field that contains the queue record ID returned by the PUT QUEUE function.

DELETE/KEEP
Specifies whether the queue record will be deleted from the queue after it is passed to the requesting program:

DELETE
Deletes the record from the queue. Note that if DELETE is specified and the record has been truncated, the truncated data is lost. DELETE is the default.

KEEP
Keeps the record in the queue.

LOCK/NOLOCK
Specifies whether the system is to retain a lock on the current queue record:

LOCK
Retains the lock on the current queue record until either a COMMIT TASK command is issued or the issuing task terminates. While a queue record is locked, no other task can access that record (regardless of its position in the queue) until the lock has been released. LOCK is the default.

NOLOCK
Releases the lock on the current queue record following the execution of a subsequent queue I/O request.

WAIT/NOWAIT
Specifies whether the issuing task is to suspend execution if the requested record cannot be found in the queue:

WAIT
Suspends task execution until the requested queue exists.

NOWAIT
Continues task execution in the event of a nonexistent queue. An ERROR_STATUS value of 4405 indicates that the requested queue record cannot be found. NOWAIT is the default.

INTO <u>(return-queue-data-location)</u>
Indicates the program variable-storage entry of the data area reserved for the requested queue record. *Return-queue-data-location* is the symbolic name of a user-defined field. The length of the data area is determined by one of the following specifications:

TO <u>(end-queue-data-location)</u>
Indicates the end of the program variable-storage entry reserved for the requested queue record and is specified following the last data-item entry in

return-queue-data-location. *End-queue-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the requested queue record.

MAX LENGTH (queue-data-max-length)
Explicitly defines the length of the data area reserved for the requested queue record. *Queue-data-max-length* is either the symbolic name of the user-defined field that contains the length of the queue record's data, or the length itself expressed as a numeric constant.

RETURN LENGTH INTO (queue-data-actual-length)
Specifies the location to which the system will return the actual length of the retrieved queue record. *Queue-data-actual-length* is the symbolic name of a user-defined 4-byte field. If the record has been truncated, the value returned to this field is the actual length of the queue record before truncation.

Example: The following example retrieves the first record in the RES_Q queue, return it to the PEND_RES field, and keep the record in the queue:

```
GET QUEUE
   ID ('RES_Q')
   FIRST
   KEEP
   INTO (PEND_RES) MAX LENGTH (125);
```

Status codes: Upon completion of the GET QUEUE function, the ERROR_STATUS field of the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4404	The requested queue header record cannot be found.
4405	The requested queue record cannot be found.
4407	An I/O error has occurred during processing.
4419	The program storage area specified for return of the queue record is too small; the returned record has been truncated as appropriate to fit the available space.
4431	The parameter list is invalid. In DC_BATCH, this code signifies that the specified record length has exceeded the maximum length based on the packet size.
4432	The derived length of the queue record data area is negative.

6.41 GET SCRATCH (DC/UCF)

The GET SCRATCH statement obtains a scratch record and places it in a storage area associated with the issuing program. The storage area must already be allocated to the requesting task; no implicit GET STORAGE function is performed during the GET SCRATCH operation. If the scratch record is larger than the designated storage area, data is truncated.

Syntax

```
►►── GET SCRATCH ─┬───────────────────────────────┬──────────────────►
                  └─ AREA ID (scratch-area-id) ──┘

►─┬──────────────────────────────────┬─┬─────────────┬──────────────►
  ├─ NEXT ← ─────────────────────────┤ ├─ DELETE ← ─┤
  ├─ FIRST ──────────────────────────┤ └─ KEEP ─────┘
  ├─ LAST ───────────────────────────┤
  ├─ CURRENT ────────────────────────┤
  ├─ PRIOR ──────────────────────────┤
  └─ RECORD ID (scratch-record-id) ──┘

►── INTO (return-scratch-data-location) ─────────────────────────────►

►─┬─ TO (end-scratch-data-location) ─────────────┬───────────────────►
  └─ MAX LENGTH (scratch-data-max-length) ───────┘

►─┬────────────────────────────────────────────────────┬─ ; ────────►◄
  └─ RETURN LENGTH INTO (scratch-data-actual-length) ──┘
```

Parameters

AREA ID (scratch-area-id)

Identifies the scratch area associated with the record being retrieved. *Scratch-area-id* is either the symbolic name of a user-defined field that contains the 1- to 8-character scratch area ID or the ID itself enclosed in single quotation marks. If AREA ID is not specified, an area ID of eight blanks is assumed.

NEXT/FIRST/LAST/CURRENT/PRIOR/RECORD ID *(scratch-record-id)*

Specifies the scratch record to be retrieved:

NEXT

Retrieves the next record in the scratch area. NEXT is the default.

FIRST

Retrieves the first record in the scratch area.

LAST

Retrieves the last record in the scratch area.

CURRENT

Retrieves the current record in the scratch area; the current record is the record most recently referenced by another scratch function.

PRIOR

Retrieves the prior record in the scratch area.

RECORD ID (scratch-record-id)
Retrieves the specified scratch record. *Scratch-record-id* is the symbolic name of a user-defined FIXED BINARY(31) field that contains the 4-byte scratch record ID.

DELETE/KEEP
Specifies whether the scratch record will be deleted from the scratch area after it is passed to the requesting program:

DELETE
Deletes the record from the scratch area. If DELETE is specified and the record has been truncated, the truncated data is lost. To maintain currency following a DELETE request, the system saves the next and prior currencies of the scratch area. DELETE is the default.

KEEP
Keeps the record in the scratch area.

INTO (return-scratch-data-location)
Specifies the program variable-storage entry of the data area to which the system will return the scratch record. *Return-scratch-data-location* is the symbolic name of a user-defined field. The length of the data area is determined by one of the following specifications:

TO (end-scratch-data-location)
Indicates the end of the data area to which the system will return the scratch record and is specified following the last data-item entry in *return-scratch-data-location*. *End-scratch-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the scratch record.

MAX LENGTH (scratch-data-max-length)
Specifies the length, in bytes, of the data area associated with the requested scratch record. *Scratch-data-max-length* is either the symbolic name of a program variable-storage field that contains the length, or the length itself expressed as a numeric constant.

RETURN LENGTH INTO (scratch-data-actual-length)
Specifies the symbolic name of the program variable-storage entry to which the system will return the actual length of the requested scratch record. If the record has been truncated, *scratch-data-actual-length* will contain the length of the full, untruncated scratch record.

Example: The following statement returns the contents of the current record in the scratch area to the variable-storage area defined by WORK_PROC_AREA and END_WORK_PROC_AREA:

```
GET SCRATCH
   CURRENT
   INTO (WORK_PROC_AREA) TO (END_WORK_PROC_AREA);
```

Status codes: Upon completion of the GET SCRATCH function, the ERROR_STATUS field of the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4303	The requested scratch area ID cannot be found.
4305	The requested scratch record ID cannot be found.
4307	An I/O error has occurred during processing.
4319	The program storage area specified for return of the scratch record is too small; the returned record has been truncated to fit the available space.
4331	The parameter list is invalid.
4332	The derived length of the scratch record is negative.

6.42 GET STORAGE (DC/UCF)

The GET STORAGE statement is used either to acquire variable storage from a system storage pool or to obtain the address of a previously acquired storage area. Once acquired, the storage is available for use:

- By the issuing task only (user storage)
- By subsequent tasks running on the same terminal (user kept storage)
- By all tasks in the system (shared or shared kept storage)

Storage availability is governed by GET STORAGE parameter specifications.

Syntax

```
►►── GET STORAGE FOR (storage-data-location) ─────────────────────────►

►─┬─ TO (end-storage-data-location) ─┬────────────────────────────────►
  └─ LENGTH (storage-data-length) ───┘

►── POINTER (storage-data-location-pointer) ─┬──────────┬─┬──────┬───►
                                             ├─ WAIT ← ─┤ └ KEEP ┘
                                             └─ NOWAIT ─┘

►─┬──────────┬─┬───────────┬─┬──────────────────────┬─────────────────►
  ├─ LONG ← ─┤ ├─ USER ← ──┤ └ STGID (storage-id) ──┘
  └─ SHORT ──┘ └─ SHARED ──┘

►─┬────────────────────────┬──────────────────────────────────────────►
  └─ VALUE (initial-value) ┘

►─┬─────────────────────────────┬─ ; ─────────────────────────────────►◄
  └─ LOCATION IS ──┬─ ANY ← ─┬──┘
                   └─ BELOW ─┘
```

Parameters

FOR (storage-data-location)
: Specifies the variable associated with the storage area being acquired. *Storage-data-location* is a user-assigned symbolic name.

TO (end-storage-data-location)
: Indicates the end of the data area for which the system will acquire storage. If this option is specified, *storage-data-location* must be declared as a PL/I structure variable. *End-storage-data-location* is the symbolic name of either a user-defined dummy byte field or a variable field not associated with the storage area. *End-storage-data-location* is specified after the last elementary data-item entry in the structure.

LENGTH (storage-data-length)
: Explicitly defines the length of the data area associated with the requested storage area. This option is specified in place of TO *(end-storage-data-location)*. If the LENGTH option is used, then no restrictions are placed on the data type; that is, *storage-data-location* does not have to be defined as a PL/I structure variable.

Storage-data-length is a user-assigned fixed binary field containing the storage length, or the length itself expressed as a numeric constant.

POINTER (storage-data-location-pointer)

Specifies the user-assigned pointer variable associated with *storage-data-location*. *Storage-data-location-pointer* is defined in variable storage with the pointer attribute. Upon successful completion of the GET STORAGE request, the system returns the address of the storage area to *storage-data-location-pointer*.

WAIT/NOWAIT

Specifies whether the issuing task is to wait for sufficient storage in the event that storage is not immediately available to meet the requirements of the GET STORAGE request:

WAIT

Specifies that the issuing task will wait until sufficient storage is available in a storage pool. WAIT is the default.

NOWAIT

Specifies that the issuing task will not wait for storage to become available if an insufficient storage condition exists. If NOWAIT is specified, the program should check the ERROR_STATUS field in the IDMS-DC communications block to determine if the GET STORAGE request has been completed. If the ERROR_STATUS value is 3202, the program should perform alternative processing before reissuing the GET STORAGE request.

KEEP

Optionally specifies whether the storage area will be used by subsequent tasks executing on the same logical terminal. When KEEP is specified, the storage area can be accessed by subsequent tasks; otherwise the storage area cannot be accessed by subsequent tasks. For a more detailed discussion of the KEEP parameter, refer to *CA-IDMS Navigational DML Programming*.

LONG/SHORT

Specifies whether the system should allocate the storage from the bottom or the top of a storage pool:

LONG

Allocates storage from the bottom of the storage pool. You should specify LONG when allocating kept storage to be held across pseudo-converses. LONG is the default.

SHORT

Allocates storage from the top of the storage pool. You should specify SHORT when allocating small pieces of storage for a short duration.

An incorrect LONG/SHORT specification will not affect normal program execution; however, it may affect the overall performance of the DC/UCF system.

USER/SHARED

Specifies whether access to the storage area is to be restricted to the issuing task or is to be available to all tasks in the system:

USER

Specifies that *only* the issuing task can access the storage area or, if KEEP is specified, only subsequent tasks executing on the same terminal. USER is the default.

Note: During system execution, a program defined at system generation with the NOPROTECT option can access any storage area within the system, including an area associated exclusively with another task. Thus, the USER attribute may not protect the storage area being acquired. However, storage areas can be protected on a system-wide or program-by-program basis during system generation and by the modes specified when storage is allocated.

SHARED

Specifies that any task in the system can access and modify the acquired storage. Each task must establish addressability to the storage area by explicitly issuing a GET STORAGE request.

STGID <u>(storage-id)</u>

Specifies the 4-character ID associated with the storage area. The STGID parameter must be specified with GET STORAGE requests for either previously allocated storage areas or areas to be reallocated. *Storage-id* is either the symbolic name of a user-defined field that contains the storage ID, or the ID itself enclosed in single quotation marks.

The specified storage ID must be unique; although multiple variable-storage areas (that is, one shared and the others user) can have the same ID, only one such area can be owned by a given task at a time. To access the IDMS-DC common work area, specify STGID 'CWA'.

Note: If the STGID parameter specifies the address of an existing storage area, the USER/SHARED parameter must specify the same option as that specified in the GET STORAGE statement that originally allocated the storage area.

VALUE <u>(initial-value)</u>

Specifies (for new storage only) the value to which the storage area will be initialized before it is returned to the issuing program. *Initial-value* specifies either the symbolic name of a user-defined field that contains the initial value or the value itself enclosed in single quotation marks. All bytes of the acquired storage area are initialized to the same value.

LOCATION IS ANY/BELOW

Specifies that storage must be allocated from below the 16-megabyte line (BELOW) or is eligible for allocation above the 16-megabyte line (ANY). ANY is the default.

Example: The following statement allocates the shared kept storage area, 09PA, and initializes it to all zeros:

```
GET STORAGE FOR (EMPLMENU_KEPT_STORAGE)
   TO (EMPLMENU_KEPT_STORAGE_END)
   NOWAIT
   KEEP
   SHORT
   SHARED
   STGID ('09PA')
   VALUE (LOW_VALUE);
```

Status codes: Upon completion of the GET STORAGE function, the ERROR_STATUS field of the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
3201	The requested storage cannot be allocated immediately; to wait would cause a deadlock.
3202	The requested storage cannot be allocated because insufficient space exists in the storage pool.
3210	The request specified a storage ID that did not previously exist; the required space has been allocated.
3231	The request specifies an invalid parameter list.
3232	The requested length is zero or negative. The request cannot be serviced because the variable storage The request cannot be serviced because the specified 01-level

6.43 GET TIME (DC/UCF)

The GET TIME statement obtains the time of day and date from the operating system. The system time is returned to the issuing task in either fixed binary, packed decimal, or edited format. The date is returned to the program in packed decimal format.

Syntax

```
►►─── GET TIME ─┬──────────────────────────────────────┬──────────────►
                └─┬─────────────┬─ INTO (return-time) ─┘
                  ├─ BINARY ← ──┤
                  ├─ DECIMAL ───┤
                  └─ EDIT ──────┘

 ►─┬──────────────────────────────┬─ ; ─────────────────────────────►◄
   └─ DATE INTO (return-date) ────┘
```

Parameters

BINARY/DECIMAL/EDIT
Specifies the format in which the time is to be returned to the issuing program. The requested formats can be fixed binary, decimal, or edited. In all cases, the returned value indicates the time since midnight:

BINARY
Returns the time in pure (absolute) binary format representing the elapsed time since midnight in ten-thousandths of a second. If BINARY is specified, the field associated with *return-time* must be a fixed binary field capable of holding a number at least as large as the number of ten-thousandths seconds in a day (864,000,000). This option provides the finest resolution of time available. BINARY is the default.

DECIMAL
Returns the time in the format *ohhmmssttttc* (padded zero, hours, minutes, seconds, ten-thousandths of a second, sign). If DECIMAL is specified, the field associated with *return-time* should be declared as FIXED DECIMAL(11).

EDIT
Returns the time as an edited character string in the format *hh:mm:ss:hh* (hours, minutes, seconds, hundredths of a second). The field size and type associated with *return-time* should be defined as CHAR(11).

INTO (return-time)
Specifies the field to which the system will return the time. *Return-time* is the symbolic name of a user-defined field to which the current time will be returned. The required field size and type depend on the requested format, as described above.

DATE INTO (return-date)
Specifies the field to which the system will return the date obtained from the operating system. *Return-date* is the symbolic name of the user-defined field to which the Julian date is returned. The Julian date is returned in FIXED DECIMAL(7) format: *0yyydddc* (padded zero, current year relative to 1900, date,

sign). For example, 0099365C would represent December 31, 1999. 0100001C would represent January 1, 2000.

Example: The following statement returns the current time and date to the CURRENT_TIME and CURRENT_DATE fields, respectively:

```
GET TIME
   EDIT INTO (CURRENT_TIME)
   DATE INTO (CURRENT_DATE);
```

Status codes: Upon completion of the GET TIME function, the only possible value in the ERROR_STATUS field of the IDMS-DC communications block is 0000.

6.44 IF

The IF statement allows the program to test for the presence of member record occurrences in a set and to determine the membership status of a record occurrence in a specified set; once the set has been evaluated, the IF statement specifies further action based on the outcome of the evaluation. For example, an IF statement might be used to determine whether a set occurrence is empty and, if it is empty, to erase the owner record.

Note: 1: DML IF statements cannot be nested within PL/I IF statements. An alternative approach is to place DML IF statements within DO...END blocks, or their equivalents.

Native VSAM users: The IF statement is not valid for sets defined with member records that are stored in native VSAM datasets.

Depending on its format, the IF statement uses set or run-unit currency. The object set occurrence of an IF statement is determined by the owner of the current record of the named set; the object record occurrence is determined by the current of run unit.

Each IF statement contains a conditional phrase and an imperative statement. When an IF is issued, the DML precompiler first generates a call to the DBMS to execute the conditional phrase; the results of the test determine whether or not the imperative statement is executed.

Syntax

```
►►─ IF ─┬───────┬─ SET (set-name) ─┬─ EMPTY ──┬─ THEN imperative-statement;─►◄
        ├─ NOT ─┤                  └─ MEMBER ─┘
        └─ ¬ ───┘
```

Parameters

IF SET (set-name) EMPTY THEN imperative-statement

Evaluates the *current owner occurrence of the named set* for the presence of member record occurrences and, depending on the outcome of the evaluation, executes the imperative statement. *Set-name* must specify a set included in the subschema.

If NOT is specified, the imperative statement is executed only if the named set has one or more member records (that is, ERROR_STATUS is 1601). If NOT is omitted, the imperative statement is executed only if the set is empty (that is, ERROR_STATUS is 0000).

IF SET (set-name) MEMBER THEN imperative-statement

Determines whether the *current record of run unit* participates as a member in any occurrence of the named set and, depending on the outcome of the evaluation, executes the imperative statement. *Set-name* must specify a set included in the subschema.

If NOT is specified, the imperative statement is executed only if the named record is not a member of the named set (that is, ERROR_STATUS is 1601). If NOT is omitted, the imperative statement is executed only if the record is a member of the set (that is, ERROR_STATUS is 0000).

Example: The following statement tests the COVERAGE_CLAIMS set for existing CLAIMS members and, if no occurrences of the CLAIMS record are found (ERROR_STATUS is 0000), moves a message to that effect to the location CLAIMS_WS:

If the current occurrence of the COVERAGE_CLAIMS set contains one or more occurrences of the CLAIMS record (ERROR_STATUS is 1601), the assignment statement is ignored and the next statement in the program is executed.

```
IF SET (COVERAGE_CLAIMS) EMPTY
   THEN CLAIMS_WS = 'NONE';
```

The following statement verifies that the EMPLOYEE record that is current of run unit is not a member of the current occurrence of the OFFICE_EMPLOYEE set before code is executed to connect the EMPLOYEE record to that set:

If the EMPLOYEE record is not a member of the OFFICE_EMPLOYEE set (ERROR_STATUS is 1601), the program performs the LINK_SET procedure. If the EMPLOYEE record is already a member of the OFFICE_EMPLOYEE set (ERROR_STATUS is 0000), the CALL statement is ignored and the next statement in the program is executed.

```
IF NOT SET (OFFICE_EMPLOYEE) MEMBER
   THEN CALL LINK_SET;
```

Status codes: Upon completion of the IF function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	Either the set is empty or the record that is current of run unit is a member of the set.
1601	Either the set is not empty or the record that is current of run unit is not a member of the set.
1606	Currency has not been established for the named set.
1608	Either an invalid set name has been specified or the current record of run unit is not a member of the named set.
1613	A current record of run unit either has not been established or has been nullified by a preceding ERASE statement.

6.45 INQUIRE MAP (DC/UCF)

The INQUIRE MAP statement is used after a map input request to accomplish one of the following actions related to the input operation:

- Move map-related information into variable storage
- Test for conditions relating to global map input operations
- Test specific map fields for the presence of the cursor
- Test for conditions relating to specific map fields

Each of these actions is discussed on the following pages.

The following rules apply to INQUIRE MAP statements:

- If any of the test conditions are requested, INQUIRE MAP must specify a statement that will be executed if the condition is found to be true.
- An INQUIRE MAP statement can specify only one field-oriented inquiry. This inquiry can be specified alone or in combination with a map-specific inquiry.

6.45.1 Moving map-related data

This version of the INQUIRE MAP statement moves one of the following map-related data items into variable storage:

- The attention ID (AID) key used
- The current cursor position (row and column)
- The entered length of a specific map input field

Syntax

```
►►─── INQUIRE MAP (map-name) ──────────────────────────────────────────►

 ►─── MOVE ──┬─ AID TO (aid-indicator) ─────────────────┬─ ; ─────────►◄
             ├─ CURSOR TO (cursor-row)  (cursor-column) ─┤
             └─ IN LENGTH FOR (field-name) TO (field-length) ─┘
```

Parameters

INQUIRE MAP <u>(map-name)</u>
Specifies the map for which the inquiry is being made. *Map-name* is the 1- to 8-character name of a map that must correspond to a map name specified in the DECLARE MAP statement, as described in Chapter 5, "DML Precompiler-Directive Statements" on page 5-1.

MOVE
Moves screen-related information to program variable storage:

AID TO <u>(aid-indicator)</u>
Returns the attention ID to the specified location in variable storage. *Aid-indicator* is the symbolic name of a 1-byte user-defined field that will be

set to the 3270 AID character received in the last map input request. The following table lists the AID characters associated with each 3270-type control key.

Key	AID character
ENTER	"'" (single quote)
CLEAR	'_' (underscore)
PF1	'1'
PF2	'2'
PF3	'3'
PF4	'4'
PF5	'5'
PF6	'6'
PF7	'7'
PF8	'8'
PF9	'9'
PF10	':'
PF11	'#'
PF12	'@'
PF13	'A'
PF14	'B'
PF15	'C'
PF16	'D'
PF17	'E'
PF18	'F'
PF19	'G'
PF20	'H'
PF21	'I'
PF22	'¢'
PF23	'.'
PF24	'<'
PA01	'%'
PA02	'>'
PA03	','

CURSOR TO (cursor-row) (cursor-column)
Returns the cursor address from the last map input function to the specified location in program variable storage. *Cursor-row* and *cursor-column* are the symbolic names of user-defined FIXED BINARY(15) fields to which the row and column cursor address will be returned.

IN LENGTH FOR (field-name) TO (field-length)
Returns the length, in bytes, of the data in the named map field to the specified location in program variable storage. *Field-name* is the name of the map field for which the length is being requested; *field-length* is the symbolic name of a user-defined fixed binary field.

Example: The following example illustrates the use of an INQUIRE MAP statement to move the 3270 AID character received in the last map input request to DC_AID_IND_V:

```
INQUIRE MAP (EMPMAPLR)
   MOVE AID TO (DC_AID_IND_V);
```

6.45.2 Testing for global map input conditions

This version of the INQUIRE MAP statement tests for one of the following global map input conditions:

- If the screen was not formatted before the input operation was performed
- If one or more input fields were truncated when transferred to variable-storage data fields
- If one or more input fields were modified on the screen before being transferred
- If one or more fields that were modified on the screen are undefined in the map being used

Syntax

```
►►── INQUIRE MAP (map-name) ──────────────────────────────────────────►

►── IF INPUT ─┬─ UNFORMATTED ─┬─ THEN imperative-statement; ──────────►◄
              ├─ TRUNCATED ───┤
              ├─ CHANGED ─────┤
              └─ EXTRANEOUS ──┘
```

Parameters

MAP (map-name)
Specifies the map for which the inquiry is being made. *Map-name* is the 1- to 8-character name of a map that must correspond to a map name specified in the DECLARE MAP statement, as described in Chapter 5, "DML Precompiler-Directive Statements" on page 5-1.

IF INPUT UNFORMATTED/TRUNCATED/CHANGED/EXTRANEOUS
Tests the outcome of the last map input request for conditions relating to the data input to the program:

UNFORMATTED
Tests whether the screen had been formatted before the input operation was performed.

TRUNCATED
Tests whether any of the map fields were truncated when transferred to variable-storage data fields.

CHANGED
Tests whether any of the map fields actually had been mapped to variable-storage data fields when the map input operation was performed.

EXTRANEOUS
Tests whether the input data stream contained any data from a field not defined to the map. If this condition is true, the undefined data field is ignored by the system.

THEN imperative-statement

Specifies the action to be taken when the test condition is true. *Imperative-statement* can be a single PL/I statement, a DML statement, or a nested block of PL/I and DML statements.

Example: The following example illustrates an INQUIRE MAP statement that tests to determine if any fields in the EMPMAPLR map have been truncated and, if so, requests that the system perform the DATA_TRUNC routine:

```
INQUIRE MAP (EMPMAPLR)
   IF INPUT TRUNCATED
     THEN CALL DATA_TRUNC;
```

6.45.3 Testing for cursor position

This version of the INQUIRE MAP statement tests a specified map field for the presence of the cursor.

Syntax

```
►►── INQUIRE MAP (map-name) ─────────────────────────────────────────►

 ►── IF CURSOR AT DFLD (field-name) THEN imperative-statement;──────►◄
```

Parameters

MAP (map-name)

Specifies the map for which the inquiry is being made. *Map-name* is the 1- to 8-character name of a map that must correspond to a map name specified in the DECLARE MAP statement, as described in Chapter 5, "DML Precompiler-Directive Statements" on page 5-1.

IF CURSOR AT DFLD (field-name)

Determines whether the cursor was in the named map field during the last map input operation. *Field-name* identifies the field within the named map to be tested.

THEN imperative-statement

Specifies the action to be taken when the test condition is true. *Imperative-statement* can be a single PL/I statement, a DML statement, or a nested block of PL/I and DML statements.

Example: The following example illustrates an INQUIRE MAP statement that tests for the presence of the cursor in the PASSED_DATA_01 data field; if the cursor is present in this field, the CHECK_2 routine is performed:

```
INQUIRE MAP (EMPMAPLR)
     IF CURSOR AT DFLD (EMP_LAST_NAME_0415)
          THEN CALL CHECK_2;
```

6.45.4 Testing for input error conditions

This version of the INQUIRE MAP statement tests:

- Whether map fields have been modified.
- Whether map fields have been erased by operator action.
- Whether map fields have been truncated.
- Whether the specified map fields are either in error (the error flag has been set on for those fields) or are correct (the error flag has been set off); this option applies only to those maps and map fields for which automatic editing is enabled.

Syntax

```
►►── INQUIRE MAP (map-name) ──────────────────────────────────────────►

 ►─ IF ┬─ CURRENT ─────────────────────────────────────────────────┬─►─
       ├─ ALL ─────────────────────────────────────────────────────┤
       ├─ NONE ────────────────────────────────────────────────────┤
       ├─ ANY ─────────────────────────────────────────────────────┤
       ├─ SOME ────────────────────────────────────────────────────┤
       ├─ ALL ─┬─ BUT ────┬─ CURRENT ──────────────────────────────┤
       │       └─ EXCEPT ─┘                                        │
       │                        ┌─────────────────────┐            │
       ├┬─ ALL ─────────────────┬↓─ DFLD (field-name) ─┴───────────┤
       │├─ NONE ────────────────┤                                  │
       │├─ ANY ─────────────────┤                                  │
       │├─ SOME ────────────────┤                                  │
       │└─ ALL ─┬─ BUT ─────┬───┘                                  │
       │        └─ EXCEPT ──┘                                      │
       │ ┌─────────────────────┐                                   │
       └↓─ DFLD (field-name) ──┴───────────────────────────────────┘

►──┬─ DATA ─┬─ YES ─────────┬─┬───────────────────────────────────────►
   │        ├─ NO ──────────┤ │
   │        ├─ ERASE ───────┤ │
   │        ├─ TRUNCATED ───┤ │
   │        ├─ IDENTICAL ───┤ │
   │        └─ DIFFERENT ───┘ │
   └─ EDIT ─┬─ ERROR ───────┬─┘
            └─ CORRECT ─────┘

 ►── THEN imperative-statement; ───────────────────────────────────────►◄
```

Parameters

MAP (map-name)
Specifies the map for which the inquiry is being made. *Map-name* is the 1- to 8-character name of a map that must correspond to a map name specified in the DECLARE MAP statement, as described in Chapter 5, “DML Precompiler-Directive Statements” on page 5-1.

IF CURRENT/ALL/NONE/ANY/SOME/ALL BUT (EXCEPT) CURRENT
Specifies the map fields to which the test applies:

CURRENT
Applies the test only to the current field; that is, the map field that was referenced in the last MODIFY MAP or INQUIRE MAP statement issued by

the program. If the last MODIFY MAP or INQUIRE MAP statement specified a field list, no currency exists.

ALL
Specifies that the test is true if all map fields meet the specified condition.

NONE
Specifies that the test is true if none of the map fields meet the specified condition.

ANY
Specifies that the test is true if one or more of the map fields meet the specified condition.

SOME
Specifies that the test is true if one or more but not all of the map fields meet the specified condition.

ALL BUT CURRENT
Specifies that the test is true if all of the map fields except for the current field meet the specified condition. The keywords BUT and EXCEPT are synonymous.

IF ALL/NONE/ANY/SOME/ALL BUT DFLD <u>(field-name)</u>
Specifies the extent to which the condition applies to the map fields.

ALL
Specifies that the test is true if all of the named map fields meet the specified condition. ALL is the default.

NONE
Specifies that the test is true if none of the named map fields meet the specified condition.

ANY
Specifies that the test is true if one or more of the named map fields meet the specified condition.

SOME
Specifies that the test is true if one or more but not all of the named map fields meet the specified condition.

ALL BUT
(Release 10.2 only) specifies that the test is true if all of the data fields except the named map fields meet the specified condition. The keywords BUT and EXCEPT are synonymous.

IF DFLD <u>(field-name)</u>
Specifies the individual map fields to which the test conditions apply. *Field-name* must be the name of a field within the named map. Multiple DFLD specifications must be separated by at least one blank.

DATA IS
Specifies the input test condition.

YES
Determines if the terminal operator entered data in the named map fields.

NO
Determines if the terminal operator did not enter data in the named map fields.

ERASE
Determines if data has been erased from the named map fields.

TRUNCATED
Determines if data has been truncated in the named map fields.

IDENTICAL
Determines whether input data is identical to the map data currently in the program's variable storage. IDENTICAL is true in either of the following cases:

- The field's modified data tag (MDT) is off. On mapin, the MDT typically is off if the user did not type any characters in the field.
- The MDT is on, but each character in the input data is exactly the same as data in variable storage, including capitalization.

DIFFERENT
Determines whether input data is different from the map data currently in the program's variable storage. DIFFERENT is true if the field's MDT is on and at least one input character differs from the data in variable storage.

EDIT
Automatic editing/error handling tests for errors in the named map fields.

Note: If the EDIT parameter is specified, automatic editing must be enabled for the map and for each of the named map fields.

ERROR
Determines if the named map fields were found to be in error during automatic editing.

CORRECT
Determines if the named map fields were found to be correct during automatic editing.

THEN <u>imperative-statement</u>
Specifies the action to be taken when the test condition is true. *Imperative-statement* can be a single PL/I statement, a DML statement, or a nested block of PL/I and DML statements.

Example: The following example determines if automatic editing has detected erroneous data in any field in the EMPMAPLR map; if so, the program modifies the map temporarily to display the erroneous fields with the bright and blinking attributes:

```
INQUIRE MAP (EMPMAPLR)
   IF ANY EDIT ERROR
      THEN MODIFY MAP (EMPMAPLR) TEMPORARY
           FOR ALL ERROR FIELDS
           ATTRIBUTES BRIGHT BLINK;
```

Status codes: Upon completion of the INQUIRE MAP function, the ERROR_STATUS field of the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4629	An invalid parameter has been passed from the program.
4641	The test condition has been found to be true. (This condition is tested for automatically by PL/I DML expansion statements.)
4644	The referenced map field is not in the specified map; a possible cause is a reference to an invalid map field subscript.
4656	The referenced map contains no data fields.

6.46 KEEP

The KEEP statement places an explicit shared or exclusive lock on a record that is current of run unit, record, set, or area. Locks placed on records through the KEEP function are maintained either for the duration of the recovery unit or until explicitly released by means of the COMMIT or FINISH statements.

Syntax:

```
►►── KEEP ─┬───────────┬─ CURRENT ─┬────────────────────────┬─ ; ────────►◄
           └ EXCLUSIVE ┘           ├ RECORD (record-name) ──┤
                                   ├ SET (set-name) ────────┤
                                   └ AREA (area-name) ──────┘
```

Parameters

CURRENT
Places an explicit shared lock on the record occurrence that is current of run unit. If the optional keyword EXCLUSIVE is specified, the current record of run unit receives an exclusive lock.

RECORD (record-name)/SET (set-name)/AREA (area-name)
Specifies that the lock will be placed on the current record of the named record type, set, or area.

Example: The following example places an exclusive lock on the current EMPLOYEE record occurrence:

```
KEEP EXCLUSIVE CURRENT RECORD (EMPLOYEE);
```

Status codes: Upon completion of the KEEP function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0606	Currency has not been established for the named record, set, or area.
0608	Either the named record or set is not in the subschema or the current record of run unit is not a member of the named set.
0610	The program's subschema specifies an access restriction that prohibits execution of the KEEP function.
0623	The named area is not in the subschema.
0626	The record to be kept has been erased.
0629	Deadlock occurred during locking of target record.

6.47 KEEP LONGTERM (DC/UCF)

The KEEP LONGTERM statement establishes longterm record locks and/or monitors access to records between tasks. Longterm database locks are used in pseudo-conversational transactions and can be shared or exclusive:

- **Longterm shared locks** allow other run units to access the locked record but prevent run units from updating the record as long as the lock is maintained.
- **Longterm exclusive locks** prevent other run units from accessing the locked record. However, run units executing on the logical terminal associated with the issuing task are not restricted from accessing the locked record. Therefore, subsequent tasks in a transaction can access the locked record and complete the database processing required by the transaction.

If a record has been locked with a KEEP LONGTERM or KEEP request, restrictions exist on the type of lock that can be placed on that record by other run units. These restrictions are based on existing locks and whether the requesting run unit is executing on the same logical terminal as the run unit that originally placed the lock on the record. The following table illustrates these restrictions.

Locks in effect	Locks allowed for other run units	Locks disallowed for other run units
Shared	Shared and longterm shared	Exclusive and longterm exclusive
Exclusive	None	Shared, exclusive, longterm shared, and longterm exclusive
Longterm shared	For all run units: shared and longterm shared For run units on the same terminal: exclusive and longterm exclusive	For run units on other terminals: exclusive and longterm exclusive
Longterm exclusive	For run units on the same terminal: shared, exclusive, longterm shared, and longterm exclusive	For run units on other terminals: shared, exclusive, longterm shared, longterm exclusive

Tasks can monitor database activity associated with a specified record during a pseudo-converse and, if desired, can place a longterm lock on the record being monitored. A subsequent task can then make inquiries about that database activity for the record and take the appropriate action.

The DC/UCF system maintains information on database activity by using five bit flags, each of which is either turned on (binary 1) or turned off (binary 0). This information is returned to the program as a numeric value. The bit assignments, the corresponding numeric value returned to the program, and a description of the associated database activity follow:

Numeric value	**Bit assignment**	**Description**
16	X'00000010'	The record was physically deleted.
8	X'00000008'	The record was logically deleted.
4	X'00000004'	The record's prefix was modified; that is, a set operation (for example, CONNECT or DISCONNECT) occurred involving the record.
2	X'00000002'	The record's data was modified.
1	X'00000001'	The record was obtained.

To determine the action or combination of actions that has occurred, you can compare the numeric value returned to the program with an appropriate constant. For example:

- If the returned value is 0, no database activity occurred for the specified record.
- If the returned value is 2, the record's data was modified.
- If the returned value is 2 or greater, the record was altered in some way.
- If the returned value is 8 or greater, the record was deleted.

The maximum possible value is 31, indicating that all the above actions occurred for the specified record.

You may prefer to monitor database activity across a pseudo-converse rather than to set longterm locks. Monitoring does not restrict access to database records, sets, or areas by other run units; however, it does enable a program to test a record for alterations made by other run units. The presence of longterm locks can prevent other run units from accessing locked records for an undesirable amount of time if, during a pseudo-converse, the terminal operator fails to enter a response. If longterm locks are used, you may want to release them at specified intervals.

▸▸ For more information regarding the use of timeout intervals, refer to *CA-IDMS System Generation*.

Syntax

```
►►── KEEP LONGTERM ( ─┬─ ALL ─────────┬─ ) ────────────────────────────────────────►
                      └─ longterm-id ─┘

►─┬─ NOTIFY CURRENT ─┬─ RECORD (record-name) ─────────────────────────────────────┬─;─►◄
  │                  ├─ SET (set-name) ──────────┤                                  │
  │                  └─ AREA (area-name) ────────┘                                  │
  ├─┬─ SHARE ─────┬─ CURRENT ─┬─ RECORD (record-name) ─┬─┬─ WAIT ← ──────┬─────────┤
  │ └─ EXCLUSIVE ─┘           ├─ SET (set-name) ───────┤ ├─ NOWAIT ──────┤         │
  │                           └─ AREA (area-name) ─────┘ └─ NODEADLOCK ──┘         │
  ├─ upgrade-specification ─────────────────────────────────────────────────────────┤
  ├─ TEST ─┬────────────────────────────────────────────────────┬──────────────────┤
  │        └─ RETURN NOTIFICATION INTO (return-location) ───────┘                  │
  └─ RELEASE ───────────────────────────────────────────────────────────────────────┘
```

Expansion of **upgrade-specification**

```
►►─┬──────────────────────────────────────────────────────────────────────────────┬─►─
   └ UPGRADE ─┬─ SHARE ─────┬─┬──────────────────────────────────────────────┬────┘
              └─ EXCLUSIVE ─┘ └ RETURN NOTIFICATION INTO (return-location) ──┘

─►─┬─────────────────┬──────────────────────────────────────────────────────────►◄
   ├─ WAIT ← ───────┤
   ├─ NOWAIT ────────┤
   └─ NODEADLOCK ────┘
```

Parameters

LONGTERM (ALL)/ (longterm-id)

Specifies the 1- to 16-character identifier that will be used in subsequent KEEP LONGTERM requests to upgrade or release a longterm lock or to make inquiries about database activity associated with the specified record. *Longterm-id* is either the symbolic name of a user-defined field that contains the longterm ID, or the ID itself enclosed in single quotation marks.

ALL is used only with the RELEASE parameter (described below) to request that the system release all longterm locks kept for the logical terminal associated with the current task.

NOTIFY CURRENT RECORD (record-name)/SET (set-name) /AREA (area-name)

Monitors database activity associated with the current occurrence of the named record type or the current record of the named set or area. When NOTIFY CURRENT is specified, the system initializes a preallocated location in the program to contain information on database activity for the specified record.

SHARE/EXCLUSIVE CURRENT RECORD (record-name)/SET (set-name)/AREA (area-name)

Specifies that the current occurrence of the named record type or the current record of the named set or area will receive a longterm shared (SHARE) or longterm exclusive (EXCLUSIVE) lock.

upgrade-specification

Upgrades a previous KEEP LONGTERM NOTIFY CURRENT request by placing a shared (SHARE) or exclusive (EXCLUSIVE) longterm lock on the record identified by *longterm-id*.

WAIT
Requests the issuing task to wait for the existing lock to be released. If the wait would cause a deadlock, the system terminates the task abnormally. WAIT is the default.

NOWAIT
Requests the issuing task not to wait for the existing lock to be released.

NODEADLOCK
Requests the issuing task to wait for the existing lock to be released, unless to do so would cause a deadlock. If the wait would cause a deadlock, the system returns control to the task.

RETURN NOTIFICATION INTO (return-location)
Returns information on database activity for that record. *Return-location* is the symbolic name of a user-defined FIXED BINARY(31) field that contains the program variable-storage entry of the data area to which the system will return the information.

TEST RETURN NOTIFICATION INTO (return-location)
Requests that the system return information on database activity associated with the record identified by *longterm-id* to a previously allocated location in the program's storage. *Return-location* is the symbolic name of a user-defined FIXED BINARY(31) field that contains the program variable-storage entry of the data area to which the system will return the information.

TEST must specify a longterm lock ID that matches the longterm lock ID specified in a previous KEEP LONGTERM NOTIFY CURRENT request.

RELEASE
Releases the longterm lock for the record identified by *longterm-id* or all record locks (ALL) owned by the logical terminal associated with the current task. RELEASE also releases the information associated with a previous KEEP LONGTERM NOTIFY request.

Example: The steps below illustrate the use of the KEEP LONGTERM statement:

1. Begin monitoring database activities for the current occurrence of the EMPLOYEE record by coding:

```
KEEP LONGTERM (KEEP_ID)
   NOTIFY CURRENT RECORD (EMPLOYEE);
```

2. Return statistics of database activities for the record identified by KEEP_ID into STAT_VALUE by coding:

```
KEEP LONGTERM (KEEP_ID) TEST RETURN NOTIFICATION
              INTO (STAT_VALUE);
```

3. Depending on the value returned to STAT_VALUE, you may want to put a longterm shared lock on the EMPLOYEE record identified by KEEP_ID by coding:

```
KEEP LONGTERM (KEEP_ID) UPGRADE SHARE;
```

4. Upon processing, release all longterm locks by coding:

```
KEEP LONGTERM (ALL) RELEASE;
```

Status codes: Upon completion of the KEEP LONGTERM function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
5101	The NODEADLOCK option has been specified; however, to wait would cause a deadlock. Control has returned to the issuing task.
5102	Unable to obtain storage for the required KEEP LONGTERM control blocks.
5105	Either the requested record type cannot be found or currency has not been established.
5113	The required area control block was not found in the DMCL.
5121	Either the requested longterm ID cannot be found or the KEEP LONGTERM request was issued by a nonterminal task.
5123	The specified area cannot be found.
5131	The parameter list is invalid.
5147	The KEEP LONGTERM area has not been readied.
5148	The run unit associated with the KEEP LONGTERM request has not been bound.
5149	The NOWAIT option has been specified; however, a wait is required.
5151	A lock manager error occurred during the processing of the KEEP LONGTERM request.
5159	An error occurred in transferring the KEEP LONGTERM request to IDMSKEEP.
5160	The requested KEEP LONGTERM lock ID was already in use with a different page group.
5161	The requested KEEP LONGTERM lock ID was already in use with a different BDKey format.

6.48 LOAD TABLE (DC/UCF)

The LOAD TABLE statement instructs the system to load a table (module or program) into the program pool.

Syntax

```
►►── LOAD TABLE (table-name) POINTER (table-location-pointer) ─┬──────────┬─;─►◄
                                                               ├─ WAIT ← ─┤
                                                               └─ NOWAIT ─┘
```

Parameters

table-name
Specifies the 1- to 8-character name of the table to be loaded. *Table* is either the symbolic name of a user-defined field that contains the table, or the name itself enclosed in single quotation marks.

POINTER (table-location-pointer)
Specifies the pointer variable for referencing the loaded table. After the table has been loaded, the pointer contains the address of the beginning of the table.

WAIT
Requests the issuing task to wait until sufficient storage becomes available. If WAIT is specified and the system encounters an insufficient storage condition, the issuing task is placed in an inactive state; when the LOAD TABLE function is completed, control returns to the issuing task according to its previously established dispatching priority. WAIT is the default.

NOWAIT
Requests the issuing task not to wait for storage to become available. If NOWAIT is specified, the system returns a value of 3402 to the ERROR_STATUS field when an insufficient storage condition exists.

Example: The following source code defines the data required for use with the LOAD TABLE request:

```
DCL STATECON_POINTER POINTER;
     DCL 1 STATECON(50) BASED (STATECON_POINTER),
           3 STATE_ABB CHAR(2),
           3 STATE_FULL CHAR(15);
```

The following statement loads the STATECON table into the program variable-storage area identified by the pointer STATECON_POINTER:

```
LOAD TABLE (STATECON)
     POINTER (STATECON_POINTER);
```

Status codes: Upon completion of the LOAD TABLE function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
3401	The requested module cannot be loaded immediately due to insufficient storage; to wait would cause a deadlock.
3402	The requested module cannot be loaded because insufficient storage exists in the program pool.
3407	The requested module cannot be loaded because an I/O error has occurred during processing.
3414	The requested module cannot be loaded because it has been defined as noncurrent and is currently in use.
3415	The requested module has been overlaid temporarily in the program pool and cannot be reloaded immediately.
3436	Either the requested program is not defined in the program definition table (PDT) and is marked out of service, or null PDEs are not specified or valid in this system.

6.49 MAP IN (DC/UCF)

The MAP IN statement requests a synchronous transfer of data from map fields on the screen to the corresponding variable-storage data fields. The MAP IN statement can also be used to transfer data from an area in variable storage that contains a 3270-like data stream to map-related variable-storage data fields; this is referred to as a native-mode data transfer.

Syntax

```
►►─── MAP IN (map-name) ──────────────────────────────────────────────►

►─┬───────────────────────────────────────────────────────────────────►─
  └─┬─ IO ─┬─────────────────────────┬────────────────────────────────
    │      └─ INPUT DATA ─┬─ YES ─┬─┘
    │                     └─ NO ──┘
    └─ NOIO DATASTREAM FROM (mapped-data-location) ─────────────────────

►─┬─────────────────────────────────────────┬──────────────────────────►
  └─────────────────────────────────────┬───┘
                                        │
  ─┬─ TO (end-mapped-data-location) ─┬──┘
   └─ MAX LENGTH (data-length) ──────┘

►─┬──────────────────────────────────┬────────────────────────────────►◄
  ├─ detail-specification ───────────┤
  └─ HEADER ─┬─────────────────────┬─┘
             ├─ PAGE (page-number) ┤
             └─ MODIFIED ──────────┘
```

Expansion of **detail-specification**

```
►►─┬──────────────────────────────────────────────────────────────────►─
   └─ DETAIL ─┬─ NEXT ← ───────────────────────────────────────────────
              ├─ FIRST ─┬──────────────────────────┬─────────────────┤
              │         └─ RETURNKEY (data-field) ─┘                 │
              ├─ KEY (key-name) ─────────────────────────────────────┤
              ├─ SEQUENCE_NUMBER (sequence-field) ─┬───────────────────────────┬┤
              │                                    └ RETURNKEY (data-field)────┘│
              └─ RETURNKEY (data-field) ────────────────────────────────────────┘

►─┬──────────────────────────────────────────┬────────────────────────►◄
  └─┬───────────────────────┬─┬────────────┬─┘
    └─ PAGE (page-number) ──┘ └─ MODIFIED ─┘
```

Parameters

map-name
: Specifies the 1- to 8-character name of a map specified by the DECLARE MAP statement, as described in Chapter 5, “DML Precompiler-Directive Statements” on page 5-1.

IO/NOIO
: Specifies the type of data transfer associated with the MAP IN request:

IO INPUT DATA YES/NO

Transfers data from map fields to variable-storage data fields that are associated with the specified map.

INPUT DATA YES/NO

Specifies whether the contents of map fields will be moved to variable-storage data fields (YES) or left unchanged (NO). This specification applies to all variable-storage data fields unless overridden by an INPUT DATA IS YES/NO clause in a previously issued MODIFY MAP request.

NOIO DATASTREAM FROM <u>(mapped-data-location)</u>

Transfers data from an area in program variable storage to the variable-storage data fields that correspond to the specified map. No terminal I/O is associated with the request.

Mapped-data-location is the symbolic name of a user-defined field that contains the program variable-storage entry of the data stream to be read by the system. The length of the data stream is determined through one of the following specifications:

TO <u>(end-mapped-data-location)</u>

Indicates the end of the program variable-storage entry that contains the data stream and is specified following the last data-item entry in *mapped-data-location*. *End-mapped-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the input data stream.

MAX LENGTH <u>(data-length)</u>

Explicitly defines the length, in bytes, of the input data stream. *Data-length* is either the symbolic name of a user-defined field that contains the length of the data stream, or the length itself expressed as a numeric constant.

detail-specification

Specifies (for pageable maps only) that the MAP IN operation is to retrieve data from a modified detail occurrence (MDT set on). The contents of all map fields in the detail occurrence are retrieved unless MODIFIED is specified for the MAP IN DETAIL statement; MODIFIED causes only modified fields to be retrieved.

►► For more information on pageable maps, see the *CA-IDMS Mapping Facility*.

NEXT

Retrieves the next sequential modified detail occurrence. An end-of-data condition (ERROR_STATUS is 4668) is returned in either of the following cases:

- No detail occurrences have been modified.
- All modified detail occurrences have been mapped in already.

NEXT is the default.

FIRST

Retrieves the first available modified detail occurrence. The optional RETURNKEY *(data-field)* clause specifies the name of a variable field in which the system stores the 4-byte key value (if any) associated with the retrieved detail occurrence. If no value is associated with the detail occurrence, the system sets *data-field* to zero. *Data-field*, which does not have to be fullword aligned, is the symbolic name of either a CHAR(4) or a FIXED BINARY(31) field that contains the key value.

Note: A value is associated with a detail occurrence by using the KEY parameter in a MAP OUT DETAIL command for that occurrence.

An end-of-data condition results if all modified data occurrences already have been mapped in.

KEY (key)

Retrieves a modified detail occurrence based on the value associated with the detail occurrence. *Key* is the name of a FIXED BINARY(31) field.

Note: A value is associated with a detail occurrence by using the KEY parameter in the MAP OUT DETAIL command for that occurrence.

A detail-not-found condition is returned in either of the following cases:

- The specified occurrence is not a modified detail occurrence.
- No detail occurrence with the specified value is found.

SEQUENCE_NUMBER (sequence-field-name)

Retrieves a detail occurrence by sequence number. Detail occurrences are built at runtime by the application program and are stored in the sequence in which they are created. *Sequence-field-name* is a FIXED BINARY(31) field.

A detail-not-found condition is returned in either of the following cases:

- The specified occurrence is not a modified detail occurrence.
- No detail occurrence with the specified value is found.

The optional RETURNKEY *(data-field)* clause specifies the name of a variable field in which the system stores the 4-byte key value (if any) associated with the retrieved detail occurrence. If no value is associated with the detail occurrence, the system sets *data-field* to zero. *Data-field*, which does not have to be fullword aligned, is the symbolic name of either a CHAR(4) or a FIXED BINARY(31) field that contains the key value.

RETURNKEY (data-field)

Performs the same operation as the NEXT clause (described previously) and specifies the name of a variable field in which the system stores the 4-byte value (if any) associated with the retrieved detail occurrence. If no value is associated with the detail occurrence, the system sets *data-field* to 0. *Data-field*, which does not have to be fullword aligned, is the symbolic name of either a CHAR(4) or a FIXED BINARY(31) field that contains the key value.

PAGE (page-number)

Specifies (for pageable maps only) the name of a variable field in which to store the current value of the $PAGE field on mapin. *Page-number* is defined as a FIXED BINARY(31) field.

MODIFIED
Specifies (for pageable maps only) that, within a modified detail occurrence, only modified fields (MDT set on) are to be retrieved in the MAP IN operation.

HEADER
Specifies (for pageable maps only) that the MAP IN operation is to retrieve the contents of data fields in the header and footer areas. The contents of all data fields in the header and footer areas are retrieved unless MODIFIED is specified for the MAP IN HEADER statement; MODIFIED causes only modified fields to be retrieved.

PAGE (page-number)
Specifies (for pageable maps only) the name of a variable field in which to store the current value of the $PAGE field on mapin. *Page-number* is defined as a FIXED BINARY(31) field.

MODIFIED
Specifies (for pageable maps only) that, within a modified detail occurrence, only modified header fields (MDT set on) are to be retrieved in the MAP IN operation.

Example: The following statement reads the EMPMAPLR map. Data values are transferred from map fields on the EMPMAPLR map to the corresponding variable-storage data fields. Subsequent commands can evaluate the input values and perform appropriate processing.

```
MAP IN (EMPMAPLR)
   INPUT DATA YES;
```

The following statement maps in the next modified detail occurrence of the EMPMAPPG map:

```
MAP IN (EMPMAPPG)
   DETAIL
   NEXT;
```

Status codes: Upon completion of the MAP IN function, the ERROR_STATUS field of the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4627	A permanent I/O error has occurred during processing.
4628	The dial-up line for the terminal has been disconnected.
4631	The map request block (MRB) contains an invalid field, indicating a possible error in the program's parameters.
4632	The derived length of the specified map input data area is zero or negative.
4633	The map load module named in the MRB cannot be found.

Status code	Meaning
4638	The specified program variable storage entry has not been allocated.
4639	The terminal being used is out of service.
4640	The NOIO option has been specified but the requested data stream cannot be found.
4642	The requested map does not support the terminal device being used.
4652	The specified edit or code table either cannot be found or is invalid for use with the named map.
4654	A data conversion error has occurred; internal map data does not match the map's data description.
4655	The user-written edit routine specified for the named map cannot be found.
4664	The requested node for a header or detail was either not present or not updated.
4668	No more modified detail occurrences require mapin.
4672	The scratch record that contains the requested detail could not be accessed (internal error).

6.50 MAP OUT (DC/UCF)

The MAP OUT statement creates or modifies detail occurrences for a pageable map or requests a transfer of data from variable-storage data fields to map fields on the terminal screen. MAP OUT can also be used to transfer data to another area in program variable storage; this is referred to as a native mode data transfer.

Syntax

```
►►─── MAP OUT (map-name) ─┬─────────────┬──────────────────────────────►
                          ├─ WAIT ← ────┤
                          └─ NOWAIT ────┘

►─┬──────────────────────────┬──────────────────────────────────────────►
  ├─ io-specification ───────┤
  └─ no-io-specification ────┘

►─┬───────────────────────────────────────────────────┬─ ; ────────────►◄
  ├── DETAIL ──┬────────────┬─┬─────────────┬──┤
  │            ├─ NEW ← ────┤ └─ KEY (key) ─┘  │
  │            └─ CURRENT ──┘                  │
  └── RESUME ──┬───────────────────────────────┬┘
               └─ PAGE ──┬─ CURRENT ← ────────┤
                         ├─ NEXT ─────────────┤
                         ├─ PRIOR ────────────┤
                         ├─ LAST ─────────────┤
                         ├─ FIRST ────────────┤
                         └─ (page-number) ────┘
```

Expansion of **io-specification**

```
►►─┬───────────────────────────────────────────────────────────────────────►─
   └ IO ← ─┬──────────────────────────────────────────────────────────┬──
           └ OUTPUT ─┬──────────────────────────────────────────────┬┘
                     └ DATA ──┬─ YES ────────┬─┬─────────┬─┬──────────┬┘
                              ├─ NO ─────────┤ └ NEWPAGE ┘ └ LITERALS ┘
                              ├─ ERASE ──────┤
                              └─ ATTRibute ──┘

─►─┬───────────────────────────────────────────────────────────────────┬──►◄
   └─ MESSAGE (message-text) ──┬─ TO (end-message-data-location) ──────┤
                               └─ LENGTH (message-data-length) ────────┘
```

Expansion of **no-io-specification**

```
►►─┬─────────────────────────────────────────────────────────────────────►─
   └─ NOIO DATASTREAM INTO (mapped-data-location) ────────────────────

─►─────────────────────────────────────────────────────────────────────►─
  ─┬─ TO (end-mapped-data-location) ──────────────────────────────────
   └─ MAX LENGTH (max-data-length) ──┘

─►─┬──────────────────────────────────────────┬───────────────────────►◄
   └─ RETURN LENGTH INTO (data-actual-length) ┘
```

Parameters

map-name
Specifies the 1- to 8-character name of a map specified by the DECLARE MAP statement, as described in Chapter 5, "DML Precompiler-Directive Statements" on page 5-1.

WAIT
Specifies that the data transfer will be synchronous. The system places the issuing task in an inactive state. When the MAP OUT operation is complete, the task resumes processing according to its established dispatching priority. WAIT is the default.

NOWAIT
Specifies that the data transfer will be asynchronous; the task will continue executing. If NOWAIT is specified, the program must issue a CHECK TERMINAL before performing any other I/O operation.

io-specification
Specifies the type of data transfer associated with the MAP OUT request. IO (the default) specifies that the data transfer is to a terminal device.

OUTPUT
Specifies (for I/O requests only) screen-display options for the data being output:

DATA
Specifies whether the variable-storage data fields are to be transmitted to the terminal. This specification applies to all variable-storage data fields unless overridden by an OUTPUT DATA clause in a previously issued MODIFY MAP request. The following options apply:

YES
Transmits the contents of variable-storage data fields to the corresponding map fields.

NO
Does not transmit the contents of variable-storage data fields to the corresponding map fields. However, if the automatic error-handling facility detects an error in any field, the system will transmit the applicable attribute bytes.

ERASE
Does not transmit the contents of variable-storage data fields and fills the corresponding map fields with null values.

ATTRIBUTE
Transmits only the attribute bytes for variable-storage data fields. Data in the record buffer is not sent to the terminal.

NEWPAGE
Activates the erase-write function; the system clears the screen and transmits both literal and variable fields to the map. If NEWPAGE is not specified, the system will write over any existing screen display without first erasing it. The keywords NEWPAGE and ERASE are synonymous.

To erase individual map fields, use the OUTPUT DATA ERASE option of the MODIFY MAP statement (described later in this chapter). To request the system to erase all screen fields and to activate the erase-write function, the MAP OUT statement must specify OUTPUT DATA ERASE NEWPAGE.

LITERALS

Transmits literal fields as well as variable-storage data fields to the terminal. If LITERALS is not specified, the system will write literal fields to the map only when a MAP OUT request specifies the NEWPAGE option.

MESSAGE <u>(message-text)</u>

Specifies (for IO requests only) the message to be displayed in the map's message area. *Message-text* is the symbolic name of a program variable-storage entry that contains the message text.

Note: The MESSAGE parameter can only be used with MAP OUT DETAIL if the $MESSAGE field is associated with the detail occurrence at map generation. To reference a message stored in the data dictionary, use the ACCEPT TEXT INTO parameter of the WRITE LOG statement (explained later in this chapter) to copy the message into *message-text*.

TO <u>(end-message-data-location)</u>

Specifies the end of the program variable-storage entry that contains the message text and is specified following the last data item in *message-text*. *End-message-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the output data stream.

LENGTH <u>(message-data-length)</u>

Defines the length, in bytes, of the message text. *Message-data-length* is either the symbolic name of a user-defined field that contains the length or the length itself expressed as a numeric constant.

no-io-specification

Transfers data from variable-storage data fields associated with the named map to another area of program variable storage; no terminal I/O is associated with the request. *Mapped-data-location* is the symbolic name of a user-defined field that contains the program variable-storage entry to which the data will be transferred.

TO <u>(end-mapped-data-location)</u>

Indicates the end of the program variable-storage entry for the output data stream and is specified following the last data-item entry in *mapped-data-location*. *End-mapped-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the output data stream.

MAX LENGTH <u>(data-length)</u>

Defines the maximum length of the output data stream. *Data-length* is either the symbolic name of the user-defined fixed binary field that contains the length of the data stream or the length itself expressed as a numeric constant.

The optional RETURN LENGTH INTO *(data-actual-length)* clause specifies the program variable-storage entry to which the system will return the length, in bytes, of the output data stream. If the data stream has been truncated, *data-actual-length* contains the length before truncation.

DETAIL
Specifies (for pageable maps only) that the MAP OUT command is to create or modify a detail occurrence, and optionally associates a numeric key value with the occurrence. For more information on pageable maps, see *CA-IDMS Mapping Facility*.

NEW
Creates a detail occurrence of a pageable map. Occurrences are displayed in the order in which they are created by the application program. NEW is the default.

CURRENT
Modifies the detail occurrence that was referenced by the most recent MAP IN DETAIL or MAP OUT DETAIL statement.

KEY <u>(key)</u>
Optionally specifies a value to be associated with the created or modified detail occurrence. The 4-byte numeric value is not displayed on the terminal screen. *Key* is the name of a FIXED BINARY(31) field that contains the db-key of the database record associated with the detail occurrence.

When the KEY parameter is used with the MAP OUT DETAIL CURRENT command, the specified value replaces the value (if any) previously associated with the detail occurrence.

RESUME PAGE
Specifies (for pageable maps only) the page of detail occurrences to be mapped out to the terminal:

CURRENT
Specifies that the current page is to be redisplayed. If no page has been displayed, the first page of the pageable map is displayed. CURRENT is the default.

NEXT
Specifies that the page that follows the current page is to be displayed. If no page follows the current page, the current page is redisplayed.

PRIOR
Specifies that the page that precedes the current page is to be displayed. If no page precedes the current page, the current page is redisplayed.

FIRST
Specifies that the first available page of detail occurrences is to be displayed.

LAST
Specifies that the page of detail occurrences with the highest available page number is to be displayed.

<u>page-number</u>
Specifies a variable field that contains the number of the page to be displayed. *Page-number* is defined as a FIXED BINARY(31) field. A page number is stored in the variable field by a preceding MAP IN PAGE *(page-number)* statement that names the same numeric variable field.

Example: The following statement writes all literal and data fields associated with the EMPMAPLR map to the terminal:

```
MAP OUT (EMPMAPLR)
   OUTPUT DATA YES
   NEWPAGE
   MESSAGE (INITIAL_MESSAGE) LENGTH (80);
```

The following statement maps out the current detail; no terminal I/O is associated with this request if the first page of the pageable map is not yet filled:

```
MAP OUT (EMPMAPPG)
   DETAIL
   KEY (DBKEY);
```

Status codes: Upon completion of the MAP OUT function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4625	The output operation has been interrupted; the operator has pressed ATTENTION or BREAK.
4626	A logical error (for example, an invalid control character) has been encountered in the output data stream.
4627	A permanent I/O error has occurred during processing.
4628	The dial-up line for the terminal has been disconnected.
4631	The map request block (MRB) contains an invalid field, indicating a possible error in the program's parameters.
4632	The derived length of the specified map output data area is zero or negative.
4633	The map load module named in the MRB cannot be found.
4638	The program variable-storage entry specified for return of the output data stream has not been allocated.
4639	The terminal being used is out of service.
4640	The NOIO option has been specified but the requested data stream cannot be found.
4642	The requested map does not support the terminal device being used.
4652	The specified edit or code table either cannot be found or is invalid for use with the named map.
4653	An error has occurred in a user-written edit routine.
4654	A data conversion error has occurred; internal map data does not match the map's data description.

Status code	Meaning
4655	The user-written edit routine specified for the named map cannot be found.
4664	There is no current detail occurrence to be updated (MAP OUT DETAIL CURRENT only). No action is taken.
4668	The amount of storage defined for pageable maps at system generation time is insufficient. No action is taken. This and subsequent MAP OUT DETAIL statements are ignored.
4672	No detail occurrence, footer, or header fields exist to be mapped out by a MAPOUT RESUME command.
4676	The first screen page has been transmitted to the terminal.
4680	The last detail for a screen was written; a map page is complete and ready to be transmitted to the terminal.

6.51 MAP OUTIN (DC/UCF)

The MAP OUTIN statement requests an output data transfer (MAP OUT) followed by an input data transfer (MAP IN). MAP OUTIN combines the functions of the MAP OUT and MAP IN requests; however, it cannot be used to perform pageable map functions or native mode data transfers. By definition, the MAP OUTIN request is synchronous; it forces the program to be conversational.

Syntax

```
►►─── MAP OUTIN (map-name) ──────────────────────────────────────────────►

►─┬─────────────────────────────────────────────────────────────────────┬─►
  └─ OUTPUT ─┬────────────────────────────┬─┬────────────┬─┬───────────┬┘
             └─ DATA ─┬─ YES ───────┬─────┘ ├─ NEWPAGE ──┤ └─ LITERALS ┘
                      ├─ NO ────────┤       └─ ERASE ────┘
                      ├─ ERASE ─────┤
                      └─ ATTRibute ─┘

►─┬──────────────────────────┬──────────────────────────────────────────►
  └─ INPUT DATA ─┬─ YES ─┬───┘
                 └─ NO ──┘

►─┬──────────────────────────────────────────────────────────────────┬─ ; ──►◄
  └─ MESSAGE (message-text) ─┬─ TO (end-message-data-location) ─┬────┘
                             └─ LENGTH (data-length) ───────────┘
```

Parameters

map-name

Specifies the 1- to 8-character name of a map specified by the DECLARE MAP statement, as described in Chapter 5, "DML Precompiler-Directive Statements" on page 5-1.

OUTPUT

Specifies screen display-options for the data being output:

DATA YES/NO/ERASE/ATTRIBUTE

Specifies whether variable-storage data fields are to be transmitted to the terminal. This specification applies to all variable-storage data fields unless overridden by an OUTPUT DATA YES/NO clause in a previously issued MODIFY MAP request.

YES

Transmits the contents of variable-storage data fields to the corresponding map fields.

NO

Does not transmit the contents of variable-storage data fields to the corresponding map fields. However, if the automatic error handling facility detects an error in any field, the system will transmit the applicable attribute bytes.

ERASE

Does not transmit the contents of variable-storage data fields and fills the corresponding map fields with null values.

ATTRIBUTE
Transmits only the attribute bytes for variable-storage data fields. Data in the record buffer is not sent to the terminal.

NEWPAGE
Activates the erase-write function; the system clears the screen and transmits both literal and variable fields to the map. If NEWPAGE is not specified, the system will write over any existing screen display without first erasing it. The keywords NEWPAGE and ERASE are synonymous.

To erase individual map fields, use the OUTPUT DATA ERASE option of the MODIFY MAP statement (described later in this chapter). To request that the system erase all screen fields and activate the erase-write function, the MAP OUT statement must specify OUTPUT DATA ERASE NEWPAGE.

LITERALS
Transmits literal fields as well as variable-storage data fields to the terminal. If LITERALS is not specified, the system will write literal fields to the map only when a MAP OUT request specifies the ERASE option.

INPUT DATA YES/NO
Specifies whether the contents of map fields will be moved to variable-storage data fields (YES) or left unchanged (NO).

This specification applies to all variable-storage data fields unless overridden by an INPUT DATA YES/NO clause in a previously issued MODIFY MAP request.

MESSAGE (message-text)
Specifies the message to be displayed in the map's message area. *Message-text* is the symbolic name of a program variable-storage entry that contains the message text. The length of the message text is determined by one of the following specifications:

TO (end-message-data-location)
Specifies the end of the program variable-storage entry that contains the message text and is specified following the last data item in *message-text*. *End-message-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the output data stream.

LENGTH (data-length)
Defines the length in bytes of the message text. *Data-length* is either the symbolic name of a user-defined field that contains the length, or the length itself expressed as a numeric constant.

Note: To reference a message stored in the data dictionary, use the ACCEPT TEXT INTO parameter of the WRITE LOG statement (described later in this chapter) to copy the message into *message-text*.

Example: The following statement erases the screen, transmits literal and variable map fields (null values), and performs a mapin operation when the operator presses an AID key:

```
MAP OUTIN (EMPMAPLR)
   OUTPUT DATA ERASE NEWPAGE
   INPUT DATA YES;
```

Status codes: Upon completion of the MAP OUTIN function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4625	The I/O operation has been interrupted; the terminal operator has pressed ATTENTION or BREAK.
4626	A logical error (for example, an invalid control character) has been encountered in the output data stream.
4627	A permanent I/O error has occurred during processing.
4628	The dial-up line for the terminal is disconnected.
4631	The map request block (MRB) contains an invalid field, indicating a possible error in the program's parameters.
4633	The map load module named in the MRB cannot be found.
4639	The terminal being used is out of service.
4642	The requested map does not support the terminal device being used.
4652	The specified edit or code table either cannot be found or is invalid for use with the named map.
4653	An error has occurred in a user-written edit routine.
4654	A data conversion error has occurred; internal map data does not match the map's data description.
4655	The user-written edit routine specified for the named map cannot be found.

6.52 MODIFY MAP (DC/UCF)

The MODIFY MAP statement requests that the system modify options in the map request block (MRB) for a map; modifications can be designated as permanent or temporary. Requested revisions can be field-specific, map-specific, or both; field-specific revisions apply to the map's variable data fields.

Note: The MODIFY MAP statement parameters used to revise predefined map and/or map data field attributes have no defaults. If a MODIFY MAP parameter is not specified, the applicable option remains set to the value specified at map generation or to the value specified in a previously issued MODIFY MAP PERMANENT statement.

Syntax

```
►►── MODIFY MAP (map-name) ──────────────────────────────────────────────►

►─┬─────────────────┬─┬────────────────────────────────────────────────┬─►
  ├─ PERMANENT ← ─┤ └─ CURSOR AT ─┬─ (cursor-row)  (cursor-column) ─┤
  └─ TEMPORARY ───┘                └─ DFLD (field-name) ─────────────┘

►─┬────────────────────────────────────┬────────────────────────────────►
  └─ WCC ─↓─────────────────────────┐  │
           │ ┌──────────────────────┘──┘
           ├─┬─ RESETMDT ─┬─┤
           │ └─ NOMDT ────┘ │
           ├─┬─ RESETKBD ─┬─┤
           │ └─ NOKBD ────┘ │
           ├─┬─ ALARM ────┬─┤
           │ └─ NOALARM ──┘ │
           ├─┬─ STARTPRT ─┬─┤
           │ └─ NOPRT ────┘ │
           └─┬─ NLCR ─┬─────┘
             ├─ 40CR ─┤
             ├─ 64CR ─┤
             └─ 80CR ─┘

►─┬──────────────────────────────────────────────────────────────────┬──►─
  └─ FOR ─┬─ ALL ─┬─ BUT ────┬─ CURRENT ─────────────────────────────┤
          │       └─ EXCEPT ─┘                                       │
          ├─── ALL ─┬─────────────┬─ FIELDS ─────────────────────────┤
          │         ├─ CORRECT ─┤                                    │
          │         └─ ERROR ───┘                                    │
          │  ┌───────────────────────────────────────────────┐       │
          ├─↓┬───────────────────────┬─ DFLD (field-name) * ─┴───────┤
          │  ├─── ALL ← ─────────────┤                               │
          │  └─ ALL ─┬─ BUT ────┬────┘                               │
          │          └─ EXCEPT ─┘                                    │
          └─ DFLD (field-name) * ────────────────────────────────────┘

►─┬──────────────────┬───────────────────────────────────────────────────►─
  ├─ BACKSCAN ─────┤
  └─ NOBACKSCAN ───┘

►─┬──────────────────────────────────────┬─┬──────────────────────────┬──►─
  └─ OUTPUT DATA ─┬─ YES ───────────────┤ └─ INPUT DATA ─┬─ YES ─┬─┘
                  ├─ NO ────────────────┤                └─ NO ──┘
                  ├─ ERASE ─────────────┤
                  └─ ATTRIBUTE ─────────┘

►─┬───────────────────────────┬──────────────────────────────────────────►─
  ├─ RIGHT ─┬─ JUSTIFY ─────┤
  └─ LEFT ──┘

►─┬──────────────────────────┬─┬─────────────────────────┬───────────────►─
  ├─ PAD (pad-character) ─┤ └─ EDIT ─┬─ ERROR ────┬─┘
  └─ NOPAD ───────────────┘          └─ CORRECT ──┘

►─┬──────────────┬─┬─────────────────────────────────────┬───────────────►─
  ├─ REQUIRED ─┤ └─ ERROR MESSAGE ─┬─ ACTIVE ─────┬─┘
  └─ OPTIONAL ─┘                   └─ SUPPRESS ───┘
```

```
►──┬──────────────────────────────────┬─ ; ──────────────────────────►◄
   │                ┌───────────────┐ │
   └─ ATTRIBUTES ─↓─┴───────────────┴─┘
                   ├─ SKIP ──────────┤
                   ├─┬ ALPHAMERIC ─┬─┤
                   │ └ NUMERIC ────┘ │
                   ├─┬ PROTECTED ──┬─┤
                   │ └ UNPROTECTED ┘ │
                   ├─┬ DISPLAY ─┬────┤
                   │ ├ DARK ────┤    │
                   │ └ BRIGHT ──┘    │
                   ├─ DETECT ────────┤
                   ├─┬ MDT ───┬──────┤
                   │ └ NOMDT ─┘      │
                   ├─┬ BLINK ───┬────┤
                   │ └ NOBLINK ─┘    │
                   ├─┬ REVERSE_VIDEO ─┬┤
                   │ └ NORMAL_VIDEO ──┘│
                   ├─┬ UNDERSCORE ───┬─┤
                   │ └ NOUNDERSCORE ─┘ │
                   ├─ NOCOLOR ─────────┤
                   └─┬ BLUE ──────┬────┘
                     ├ RED ───────┤
                     ├ PINK ──────┤
                     ├ GREEN ─────┤
                     ├ TURQUOISE ─┤
                     ├ YELLOW ────┤
                     └ WHITE ─────┘
```

*Expansion of * in DFLD (field-name) **

```
*    (─┬────────────┬─field-name─┬───────────────┬─)
       └─ group-id ─┘            └─(subscript)───┘
```

Parameters

map-name

Specifies the 1- to 8-character name of a map specified by the DECLARE MAP statement, as described in Chapter 5, "DML Precompiler-Directive Statements" on page 5-1.

PERMANENT

Specifies that modifications will apply to all mapping mode I/O requests issued until the program terminates or until a subsequent MODIFY MAP request overrides the requested revisions. PERMANENT is the default.

TEMPORARY

Specifies that modifications will apply only to the next mapping mode I/O request (that is, MAP IN, MAP OUT, or MAP OUTIN).

CURSOR AT

Identifies the screen location at which the cursor will be positioned during output operations.

cursor-row cursor-column

Specifies a row and column on the terminal screen to which the cursor will be moved. *Cursor-row* is either the symbolic name of a FIXED BINARY(15) field that contains the row value or the value itself expressed as a numeric constant. *Cursor-column* is either the symbolic name of a FIXED

BINARY(15) field that contains the column value or the value itself expressed as a numeric constant.

DFLD (field-name)
Specifies that the cursor will be moved to the first position in the specified field. *Field-name* must be the name of a map field.

WCC
Specifies the write-control-character (WCC) options requested for the output operation.

Note: If a MODIFY MAP request alters any WCC option, the system resets unspecified options to the following values:

- NOMDT
- NOKBD
- NOALARM

RESETMDT/NOMDT
Specifies whether the modified data tags (MDTs) for the map fields will be reset (turned off) automatically when the map is displayed. When NOMDT is in effect, the associated data is retransmitted to variable-storage data fields during the next MAP IN request.

RESETKBD/NOKBD
Specifies whether the keyboard will (RESETKBD) or will not (NOKBD) be unlocked automatically when the map is displayed.

ALARM/NOALARM
Specifies whether the terminal audible alarm (if installed) will sound automatically when the map is displayed.

STARTPRT/NOPRT
Specifies (for 3280-type printers only) whether the contents of the terminal buffer will be printed automatically when the data has been transmitted to the terminal.

NLCR/40CR/64CR/80CR
Specifies the characters-per-line formatting for 3280-type printer output and is meaningful only if the STARTPRT option has been specified.

NLCR
Specifies that no line formatting will be performed on the printer output. Printing will begin on a new line only if the printer encounters new line (NL) and carriage control (CR) characters.

40CR
Specifies that the contents of the 3280-type printer buffer will be printed at 40 characters per line.

64CR
Specifies that the contents of the 3280-type printer buffer will be printed at 64 characters per line.

80CR
Specifies that the contents of the 3280-type printer buffer will be printed at 80 characters per line.

FOR
Specifies the map fields to be modified or excluded from modification

ALL BUT CURRENT
Modifies all fields except the current field. The current field is the map field that was referenced in the last MODIFY MAP or INQUIRE MAP request issued by the program. However, if that request referenced a list of fields rather than a single map field, no currency exists and all map fields are modified.

ALL CORRECT/ERROR FIELDS
Modifies either all fields found to be correct or all fields found to be in error during automatic editing or by a user-written edit module.

If either ALL CORRECT FIELDS or ALL ERROR FIELDS is specified, automatic editing must be enabled for the map.

ALL/ALL BUT DFLD <u>(field-name)</u>
Explicitly specifies the fields to be modified or excluded from modification. DFLD *(field-name)* names the map fields to be modified or excluded from modification. *Field-name* must be a map field. Multiple DFLD specifications come from only one record and must be separated by at least one blank. Field names that are not unique within the program must be qualified with the name of the associated record. Likewise, multiply-occurring fields must be qualified with the appropriate subscripts. Multiple DFLDs are separated by at least one blank (for example, HOSPITAL_CLAIM.DIAGNOSIS_0430(1) HOSPITAL_CLAIM.DIAGNOSIS_0430(2) HOSPITAL_CLAIM.DIAGNOSIS_0430(3)).

ALL
Specifies that all named map fields will receive the requested modifications. ALL is the default.

ALL BUT
Specifies that all map fields except those named will receive the requested modifications.

BACKSCAN/NOBACKSCAN
Indicates whether the system is to backscan the specified fields to remove trailing blanks before performing a mapout operation. If BACKSCAN is specified, only characters up to the last nonblank will be sent to the terminal; fields remaining on the screen will contain whatever characters were present before the MAP OUT or MAP OUTIN request was issued. If the MAP OUT or MAP OUTIN request specifies the ERASE option, the system erases the contents of all terminal data fields.

OUTPUT DATA YES/NO/ERASE/ATTRIBUTE
Specifies whether map fields will be set to the value of the corresponding variable-storage data fields (YES), left unchanged (NO), or erased (ERASE),

or whether only the attribute byte (ATTRIBUTE) is transmitted during an output operation.

INPUT DATA YES/NO
Specifies whether map fields will be moved automatically to the corresponding variable-storage data fields during an input operation.

RIGHT/LEFT JUSTIFY
Indicates whether the variable-storage fields should be right- or left-justified on input.

PAD <u>(pad-character)</u>/NOPAD
Indicates whether variable-storage data fields will be padded on input.

PAD <u>(pad-character)</u>
Pads the field on the right (if right justified) or left (if left justified) with the specified character. *Pad-character* can be the symbolic name of the field (CHAR(1)) containing the pad character, or the pad character itself enclosed in single quotation marks.

NOPAD
Does not pad the fields.

EDIT ERROR/CORRECT
Explicitly sets the error flag on (ERROR) or off (CORRECT) for the specified map fields. If this parameter is specified, automatic editing must be enabled for the map.

The ability to set the error flag enables programs to perform their own editing and validation in addition to that provided by the automatic editing feature. On a MAPOUT operation, if any field is flagged to be in error, then for all fields (both correct and incorrect) only attribute bytes are transmitted; no data is moved from program variable storage to the screen.

REQUIRED/OPTIONAL
Indicates whether the terminal operator will be required to enter data in the specified map fields. An error results on mapin if REQUIRED is specified and the terminal operator fails to enter data in a required field.

If this parameter is specified, automatic editing must be enabled for the map and for the specified map fields.

ERROR MESSAGE ACTIVE
Enables display of the error message associated with the field. Typically, you enable display of an error message only after specifying ERROR MESSAGE SUPPRESS for the map in a previous MODIFY MAP PERMANENT statement.

ERROR MESSAGE SUPPRESS
Disables display of the error message associated with the field. When the map is redisplayed because of errors, the error message defined for the map field will not be displayed even if the field contains edit errors.

Use of this parameter allows you flexibility in handling error messages. For instance, you can code a data validation module to suppress a map field's default error message to enable a different error message to be displayed for that field.

ATTRIBUTES
Indicates the 3270- and 3279-type terminal display attributes for the specified map fields. If multiple attributes are specified, they must be separated by at least one blank. Only the named attributes will be modified in the map's MRB.

SKIP
Indicates that the cursor will be repositioned automatically over the map fields to the next unprotected field. If SKIP is specified, the specified map fields are assigned the NUMERIC and PROTECTED attributes (described below) automatically.

ALPHAMERIC/NUMERIC
Indicates whether the data input to the map fields by the terminal operator can be alphanumeric (any character on the 3270 keyboard) or numeric. If the terminal does not have the numeric lock option, a specification of NUMERIC is ignored.

PROTECTED/UNPROTECTED
Indicates whether the specified map fields will be protected from data entry or will be available for data entry or modification by the terminal operator. UNPROTECTED cannot be specified if SKIP has been specified.

DISPLAY/DARK/BRIGHT
Indicates whether the specified map fields will be displayed in normal (DISPLAY) or bright (BRIGHT) intensity or will not be displayed (DARK). DARK cannot be specified if DETECT has been specified.

DETECT
Indicates whether the specified map fields will be detectable by a light pen. All fields assigned the BRIGHT attribute are automatically detectable by a light pen.

MDT/NOMDT
Indicates whether the modified data tag will (MDT) or will not (NOMDT) be set automatically for the map fields when displayed.

BLINK/NOBLINK
Indicates (3279s only) whether the specified map fields will be displayed with blinking characters.

REVERSE_VIDEO/NORMAL_VIDEO
Indicates (3279s only) whether the specified map fields will be displayed in reverse video (background and character colors reversed) or in normal video.

UNDERSCORE/NOUNDERSCORE
Indicates (3279s only) whether the specified map fields will be displayed with underlined characters.

NOCOLOR
Specifies (for 3279s only) that the map fields will not be displayed with color attributes.

BLUE/RED/PINK/GREEN/TURQUOISE/YELLOW/WHITE
Indicates (3279s only) that the specified map fields will be displayed with one of the seven available color attributes.

Note: UNDERSCORE, REVERSE_VIDEO, and BLINK are mutually exclusive; that is, they can be specified in conjunction with other attributes but cannot be specified with each other. For example, neither REVERSE_VIDEO nor UNDERSCORE can be assigned to a field for which the BLINK attribute has been defined.

Example: The following statement positions the cursor at EMP_ID_0415 and prohibits the terminal operator from entering data in any field except EMP_ID_0415 and DEPT_ID_0415:

```
MODIFY MAP (EMPMAPLR) TEMPORARY
   CURSOR AT DFLD (EMP_ID_0415)
   FOR ALL BUT DFLD (EMP_ID_0415) DFLD (DEPT_ID_0415)
   ATTRIBUTES PROTECTED;
```

The following statement sets the edit flag on for the TASK_CODE_01 field, thereby overriding automatic editing and error handling for the next mapin request:

```
MODIFY MAP (EMPMAPLR) TEMPORARY
   FOR DFLD (TASK_CODE_01)
   EDIT ERROR;
```

Status codes: Upon completion of the MODIFY MAP function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4629	An invalid parameter has been passed from the program.
4644	The map field is not in the specified map; a possible cause is a reference to an invalid map field subscript.
4656	The referenced map contains no data fields.

6.53 MODIFY RECORD

The MODIFY RECORD statement replaces element values of the specified record occurrence in the database with new element values defined in program variable storage.

Steps before using MODIFY RECORD: Before executing the MODIFY RECORD statement, satisfy the following conditions:

- Ready all areas affected either implicitly or explicitly in one of the update usage modes (see 6.61, “READY” on page 6-164 later in this chapter).
- Establish the specified record as current of run unit. If the record that is current of run unit is not an occurrence of the specified record, an error condition results.
- The values of all elements defined for the specified record in the program's subschema view must be in variable storage. If the MODIFY RECORD statement is not preceded by an OBTAIN statement, you must initialize the appropriate values. The best practice, however, is to precede MODIFY RECORD with an OBTAIN statement to ensure that all the elements in the modified record are present in variable storage.

Modifying CALC- and sort-control elements: The following special considerations apply to modification of CALC- and sort-control elements:

- If modification of a CALC- or sort-control element will violate a duplicates-not-allowed option, the record is not modified and an error condition results.
- If a CALC-control element is modified, successful execution of the MODIFY RECORD statement enables the record to be accessed on the basis of its new CALC-key value. The db-key of the specified record is not changed.
- If a sort-control element is to be modified, the sorted set in which the specified record participates must be included in the subschema invoked by the program. A record occurrence that is a member of a set not defined in the subschema can be modified *if the undefined set is not sorted.*
- If any of the modified elements in the specified record are defined as sort-control elements for any set occurrence in which that record is currently a member, the set occurrence is examined. If necessary, the specified record is disconnected and reconnected in the set occurrence to maintain the set order specified in the schema.

Considerations for native VSAM users: The following special considerations apply to the modification of records in native VSAM datasets:

- The length of a record in an entry-sequenced dataset (ESDS) cannot be changed even if the records are variable length.
- The prime key for a key-sequenced dataset (KSDS) cannot be modified.

Currency: The specified record must be established as current of run unit.

Following successful execution of the MODIFY RECORD statement, the modified record becomes the current record of run unit, its record type, its area, and all sets in which it participates as member or owner.

Syntax

```
►►── MODIFY RECORD (record-name); ─────────────────────────────────────►◄
```

Parameter

record-name
Defines the named record occurrence, as specified in program variable storage. *Record-name* must specify a record type included in the subschema.

Example: The following example illustrates the steps involved in modifying an occurrence of the EMPLOYEE record. Assume that the employee address is to be changed.

1. Retrieve the desired EMPLOYEE record, moving its contents to variable storage:

```
EMP_ID_0415 = EMP_ID_IN;
OBTAIN CALC RECORD (EMPLOYEE);
```

2. Update the value of the EMP_ADDRESS_0415 field by moving the new address into the proper location in the EMPLOYEE record:

```
EMP_ADDRESS_0415 = NEW_ADDRESS;
```

3. Issue a MODIFY RECORD statement to return all data items in the EMPLOYEE record to the database:

```
MODIFY RECORD (EMPLOYEE);
```

Status codes: Upon completion of the MODIFY RECORD function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0804	The OCCURS DEPENDING ON item is less than 0 or greater than the maximum number of occurrences of the control element.
0805	Modification of the record would violate a duplicates-not-allowed option for a CALC record, a sorted set, or an index set.
0806	Currency has not been established for the named record.
0808	The specified record cannot be found. The record name has probably been misspelled.
0809	The named record's area has not been readied in one of the update usage modes.

Status code	Meaning
0810	The subschema specifies an access restriction that prohibits modification of the named record.
0811	There is insufficient space to hold the modified variable-length record occurrence.
0813	A current record of run unit has not been established or has been nullified by a previous ERASE statement.
0818	The record has not been bound.
0820	The current record of run unit is not the same type as the named record.
0821	An area other than the area of the named record has been readied with an incorrect usage mode.
0825	No current record of set type has been established.
0833	At least one sorted set in which the named record participates has not been included in the subshema.
0855	An invalid length has been defined for a variable length record.
0860	A record occurrence has been encountered whose type is inconsistent with the set named in the ERROR_SET field of the IDMS-DB communications block; probable causes include: a broken chain and improper database description.
0883	Either the length of a record in a native VSAM ESDS has been changed or a prime key in a native VSAM KSDS has been modified.

6.54 MODIFY RECORD (LRF)

The MODIFY RECORD statement changes field values in an existing logical-record occurrence. LRF uses the field values present in the variable-storage location reserved for the logical record to update the appropriate database records. You can optionally specify an alternative variable storage location from which the changed field values are to be obtained.

Syntax

```
►►── MODIFY RECORD (logical-record-name) ──────────────────────────────►

►─┬────────────────────────────────────────┬─┬─────────────────────────────────┬─►
  └─ FROM (alt-logical-record-location) ───┘ └─ WHERE (boolean-expression) ────┘

►─┬───────────────────────────────────────────────────┬─ ; ───────────►◄
  └─ ON LR_STATUS (path-status)  imperative-statement ┘
```

Parameters

logical-record-name
: Defines the named logical-record occurrence, as specified in program variable storage. Unless the FROM clause is specified (see below), the field values used to update the database are taken from the area in program variable storage reserved for the named logical record. *Logical-record-name* must specify a logical record defined in the subschema.

FROM (alt-logical-record-location)
: Names an alternative variable-storage location from which the field values used to perform the requested modification are to be obtained. When modifying a logical record that was retrieved into an alternative location in variable storage, the FROM clause should name the same location specified in the OBTAIN request. If the FROM clause is included in the MODIFY RECORD statement, *alt-logical-record-location* must identify a record location defined in program variable storage.

WHERE boolean-expression
: Specifies the selection criteria to be applied to the named logical record. For details on coding the WHERE clause, see 6.78, "Logical-record clauses (WHERE and ON)" on page 6-215 at the end of this chapter.

ON LR_STATUS (path-status) imperative-statement
: Specifies the action to be taken if *path-status* is returned to the LR_STATUS field in the LRC block. *Path-status* must be a 1- to 16-character alphanumeric value. For details on coding this clause, see 6.78, "Logical-record clauses (WHERE and ON)" on page 6-215 at the end of this chapter.

Example: The following example illustrates the steps taken to modify an occurrence of the EMP_SKILL_LR logical record. Assume that the skill level for employee 120 is to be upgraded from 02 (COMPETENT_0425) to 03 (PROFICIENT_0425).

1. Retrieve the desired logical-record occurrence:

```
OBTAIN FIRST RECORD (EMP_SKILL_LR)
    WHERE (EMP_ID_0415 = '0120'
      AND SKILL_ID_0455 = '3610'
      AND SKILL_LEVEL_0425 = '02');
```

2. Update the SKILL_LEVEL_0425 field:

   ```
   SKILL_LEVEL_0425 = '03';
   ```

3. Issue the MODIFY RECORD (LRF) statement for the updated EMP_SKILL_LR logical record:

   ```
   MODIFY RECORD (EMP_SKILL_LR);
   ```

LRF retrieves the EMP_SKILL_LR logical record where EMP_ID_0415 = '0120', SKILL_ID_0455 = '3610', and SKILL_LEVEL_0425 = '02'. The EXPERTISE occurrence represents the only data physically modified in the database.

```
EMP_SKILL_LR

EMPLOYEE        EXPERTISE       SKILL
120             04              7620
120             03              3710
120             02 (now 03)     3610
```

6.55 OBTAIN (LRF)

The OBTAIN statement retrieves the named logical record and places it in the variable-storage location reserved for that logical record. The OBTAIN statement can be issued to retrieve a single logical record, or it can be issued in iterative logic to retrieve all logical records that meet criteria specified in the WHERE clause. Additionally, the OBTAIN statement can specify that the retrieved logical record is to be placed into an alternative variable storage location.

Syntax

```
►►─── OBTAIN ─┬─────────┬── RECORD (logical-record-name) ──────────────►
              ├─ FIRST ─┤
              └─ NEXT ← ┘

►─┬──────────────────────────────────────┬─┬─────────────────────────────┬─►
  └─ INTO (alt-logical-record-location) ─┘ └─ WHERE (boolean-expression) ─┘

►─┬──────────────────────────────────────────────────┬─ ; ────────────►◄
  └─ ON LR_STATUS (path-status)  imperative-statement ─┘
```

Parameters

FIRST

Retrieves the first occurrence of the logical record. OBTAIN FIRST is typically used to retrieve the first in a series of logical-record occurrences following the iterative retrieval of a different series of logical-record occurrences.

NEXT

Retrieves a (subsequent) occurrence of the named logical record, in the order specified by the DBA in the path. OBTAIN NEXT is typically issued in iterative logic to retrieve a series of logical-record occurrences (possibly including the first).

When LRF receives repeated OBTAIN NEXT commands, it replaces field values in program variable storage with new values obtained through repeated access to the appropriate database records, thereby supplying the program with new occurrences of the desired logical record.

If an OBTAIN FIRST statement is followed by an OBTAIN NEXT statement to retrieve a series of occurrences of the same logical record, the OBTAIN statements must direct LRF to the same path. For this reason, you must ensure that the selection criteria specified in the WHERE clause that accompanies the OBTAIN FIRST and OBTAIN NEXT statements describe the same attributes of the desired logical record.

If the program issues an OBTAIN NEXT statement without issuing an OBTAIN FIRST, or if the last path status returned for the path was LR_NOT_FOUND, LRF interprets the OBTAIN NEXT as OBTAIN FIRST. After LR_ERROR or a DBA-defined path status, LRF does *not* interpret OBTAIN NEXT as OBTAIN FIRST.

RECORD <u>(Logical-record-name)</u>
Defines the named logical record occurrence, as specified in program variable storage. *Logical-record-name* must specify a logical record defined in the subschema.

INTO <u>(alt-logical-record-location)</u>
Specifies an alternative location in variable storage into which LRF will place the retrieved logical record. Any subsequent MODIFY, STORE, or ERASE statements for a logical record placed in *alt-logical-record-location* should name that area as the one from which LRF will obtain the data to be used to update the logical record.

WHERE <u>(boolean-expression)</u>
Specifies the selection criteria to be applied to the named logical record. For details on coding this clause, see 6.78, "Logical-record clauses (WHERE and ON)" on page 6-215 at the end of this chapter.

ON LR_STATUS <u>(path-status) imperative-statement</u>
Specifies the action to be taken if *path-status* is returned to the LR_STATUS field in the LRC block. *Path-status* must be a 1- to 16-character alphanumeric value. For details on coding this clause, see 6.78, "Logical-record clauses (WHERE and ON)" on page 6-215 at the end of this chapter.

Example: The following example illustrates the use of the OBTAIN NEXT statement to retrieve a series of logical-record occurrences. The program issues the OBTAIN NEXT statement iteratively to retrieve the first and all subsequent occurrences of the EMP_JOB_LR logical record for all employees in the specified department.

```
GET_AN_ORDER: PROC OPTIONS(MAIN);
  DEPT_ID_0410 = DEPT_ID_IN;
  OBTAIN NEXT RECORD (EMP_JOB_LR)
    WHERE (DEPT_ID_0410 = DEPT_ID_0410 OF LR);
     IF LR_STATUS = 'LR_ERROR' THEN
       CALL ERROR_PROCESSING;
     IF LR_STATUS = 'LR_NOT_FOUND' THEN
       CALL END_PROCESSING;
     .
     .
     .
  GO TO GET_AN_ORDER;
END GET_AN_ORDER;
```

The following figure illustrates the information retrieved by each OBTAIN NEXT statement.

	DEPARTMENT	EMPLOYEE	OFFICE	JOB
ONE OCCURRENCE OF EMP_JOB_LR	5100	466	8	SNOWBLOWER
	5100	467	8	WINDKEEPER
	5100	334	5	RAINDANCE
	5100	457	8	STURM UND DRANG

The EMP_JOB_LR logical record consists of DEPARTMENT, OFFICE, EMPLOYEE, and JOB information.

6.56 POST (DC/UCF)

The POST statement alters an event control block (ECB) either by posting it to indicate completion of an event upon which another task is waiting, or by clearing it to an unposted status.

Note: Programs posting and waiting on ECBs are responsible for clearing ECBs before issuing subsequent WAIT requests.

Syntax

```
►►── POST ─┬─ EVENT (ecb-name) ─────────────────┬─ ; ──────────────────────►◄
           └─ EVENT NAME (ecb-id) ─┬─────────┬──┘
                                   └─ CLEAR ─┘
```

Parameters

EVENT (ecb)
Identifies the ECB to be posted. *Ecb* is the symbolic name of a user-defined area composed of three binary fullword fields that contain the ECB. Program-allocated ECBs are cleared by setting *ecb* to zero.

EVENT NAME (ecb-id)
Specifies the 4-character symbolic ID of the ECB to be posted or cleared. *Ecb-id* is either the symbolic name of a user-defined field that contains the ECB ID, or the ID itself enclosed in single quotation marks.

CLEAR
Specifies that the ECB identified by *ecb-id* is cleared to an unposted status.

Example: The following example posts the event whose ECB identifier is in the FOUND_ECB field and to clear the ECB to an unposted status:

```
POST
   EVENT NAME (FOUND_ECB)
   CLEAR;
```

Status codes: Upon completion of the POST function, the only possible value in the ERROR_STATUS field of the IDMS-DC communications block is 0000.

6.57 PUT QUEUE (DC/UCF)

The PUT QUEUE statement stores a queue record in either the DDLDCRUN or the DDLDCQUE area of the data dictionary. The DC/UCF system assigns an ID to the queue record and places it at the beginning or end of its associated queue.

Syntax

```
►►── PUT QUEUE ─┬───────────────────┬─┬──────────┬────────────────────────►
                └─ ID (queue-id) ───┘ ├─ FIRST ──┤
                                      └─ LAST ← ─┘

►── FROM (queue-data-location) ─┬─ TO (end-queue-data-location) ─┬───────►
                                └─ LENGTH (queue-data-length) ───┘

►─┬────────────────────────────────────────────────────┬─────────────────►
  └─ RETURN RECORD ID INTO (return-queue-record-id) ───┘

►─┬───────────────────────────────────────────┬─ ; ─────────────────────►◄
  └─ RETENTION (queue-retention-period) ──────┘
```

Parameters

ID (queue-id)
Directs the queue record to a previously defined queue. *Queue-id* is either the symbolic name of a user-defined alphanumeric field that contains the 1- to 16-character ID, or the ID itself enclosed in single quotation marks. If a queue ID is not specified, a null ID of 16 blanks is assumed.

FIRST/LAST
Specifies whether the queue record is to be placed at the beginning or end of the queue. The default is LAST.

FROM (queue-data-location)
Specifies the program variable-storage entry associated with the data to be stored in the queue record. *Queue-data-location* is the symbolic name of a user-defined field.

TO (end-queue-data-location)
Indicates the end of the program variable-storage entry that contains the data to be stored in the queue and is specified following the last data-item entry in *queue-data-location*. *End-queue-data-location* is the symbolic name of a user-defined dummy byte field or a field that contains a data item not associated with the queue record.

LENGTH (queue-data-length)
Explicitly defines the length, in bytes, of the area that contains the data to be stored in the queue record. *Queue-data-length* is either the symbolic name of a user-defined field that contains the length or the length itself expressed as a numeric constant.

RETURN RECORD ID INTO (return-queue-record-id)
Specifies the location in the program to which the system will return the system assigned ID of the queue record. *Return-queue-record-id* is the symbolic name of

a user-defined FIXED BINARY(31) field. The returned ID is used to reference the queue record in subsequent GET QUEUE and DELETE QUEUE statements.

RETENTION (queue-retention-period)

Specifies the time, in days, that the system will retain the queue in the data dictionary. At system startup, queues having expired retention periods are deleted automatically by the system. The retention period begins when the first record is stored in the queue.

Queue-retention-period is either the symbolic name of a user-defined fixed binary field that contains the retention period or the retention period itself expressed as a numeric constant in the range 0 through 255. A retention period of 255 indicates that the queue is never to be deleted automatically by the system. The specified retention period takes precedence over retention periods associated with previously defined queues. The RETENTION parameter is ignored if the record being allocated is not the first record in the queue.

Example: The following example allocates a queue record in the beginning of the RES_Q queue, return the ID of the record to the Q_REC_ID field, and retain the queue for 45 days:

```
PUT QUEUE
   ID ('RES-Q')
   FIRST
   FROM (NEW_RES) TO (END_NEW_RES)
   RETURN RECORD ID INTO (Q_REC_ID)
   RETENTION (45);
```

Status codes: Upon completion of the PUT QUEUE function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4407	Either the queue upper limit was reached or an I/O error occurred during processing.
4431	The parameter list is invalid; under DC-BATCH, this status indicates that the specified record length exceeds the maximum length based on the packet size.
4432	The derived length of the specified queue record is either zero or negative.

6.58 PUT SCRATCH (DC/UCF)

The PUT SCRATCH statement stores or replaces a scratch record in the DDLDCSCR area of the data dictionary. For new records, PUT SCRATCH generates an index entry in a scratch area associated with the issuing task. If the scratch area does not already exist, the system allocates it dynamically in the storage pool.

Syntax

```
►►── PUT SCRATCH ─┬─────────────────────────────┬─────────────────────────►
                  └─ AREA ID (scratch-area-id) ─┘

►── FROM (scratch-data-location) ─┬─ TO (end-scratch-data-location) ─────┬─►
                                  └─ LENGTH (scratch-data-location-length)┘

►─┬──────────────────────────────────────────────────┬─────────────────────►
  └─ RECORD ID (scratch-record-id) ─┬─────────────┬──┘
                                    └─ REPLACE ───┘

►─┬───────────────────────────────────────────────────┬─ ; ──────────────►◄
  └─ RETURN RECORD ID INTO (scratch-record-id) ───────┘
```

Parameters

AREA ID <u>(scratch-area-id)</u>
Specifies the 1- to 8-character ID of the scratch area associated with the record being allocated. *Scratch-area-id* is either the symbolic name of a user-defined field that contains the ID or the ID itself enclosed in single quotation marks. If AREA ID is not specified, an area ID of eight blanks is assumed.

FROM <u>(scratch-data-location)</u>
Specifies the data to be stored in the scratch record. *Scratch-data-location* is the symbolic name of a user-defined program variable-storage entry that contains the data.

TO <u>(end-scratch-data-location)</u>
Indicates the end of the data area to be stored in the scratch record and is specified following the last data-item entry in *scratch-data-location*. *End-scratch-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the scratch data being stored.

LENGTH <u>(scratch-data-location-length)</u>
Defines the length, in bytes, of the data area. *Scratch-data-location-length* is the symbolic name of a user-defined field that contains the length or the length itself expressed as a numeric constant.

RECORD ID <u>(scratch-record-id)</u>
Specifies the ID of the scratch record being stored. *Scratch-record-id* is either the symbolic name of a user-defined FIXED BINARY(31) field that contains the ID or the ID itself expressed as a numeric constant.

REPLACE
Specifies that the scratch record identified by *scratch-record-id* replaces an existing scratch record. If REPLACE is specified and the scratch record identified

by *scratch-record-id* does not exist, the record is stored and a status value of 0000 is returned.

RETURN RECORD ID INTO (scratch-record-id)

Requests that the system return the automatically assigned ID of a scratch record to the program. *Return-scratch-record-id* is the symbolic name of a user-defined field into which the system will place the 4-byte scratch record ID.

Example: The following statement replaces the scratch record identified by SCR_REC_ID with data in the WORK_PROC_AREA field:

```
PUT SCRATCH
   FROM (WORK_PROC_AREA) LENGTH (125)
   RECORD ID (SCR_REC_ID) REPLACE;
```

Status codes: Upon completion of the PUT SCRATCH function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request to add a scratch record has been serviced successfully.
4305	The requested scratch record ID cannot be found.
4307	An I/O error has occurred during processing.
4317	The request to replace a scratch record has been serviced successfully.
4322	The request to add a scratch record cannot be serviced because the specified scratch record already exists in the scratch area and REPLACE has not been specified.
4331	The parameter list is invalid.
4332	The derived length of the specified scratch record is either zero or negative.

6.59 READ LINE FROM TERMINAL (DC/UCF)

The READ LINE FROM TERMINAL statement requests a synchronous, line-by-line transfer of data from the terminal to the issuing program.

Syntax

```
►►── READ LINE FROM TERMINAL ─┬──────┬─┬───────────┬──────────────────────►
                              └ ECHO ┘ └ NOBACKPAGE ┘

 ►─ INTO (input-data-location) ┬ TO (end-input-data-location) ──────────────►
                               └ MAX LENGTH (input-data-location-max-length)┘

 ►┬──────────────────────────────────────────────┬ ; ─────────────────────►◄
  └ RETURN LENGTH INTO (input-data-actual-length) ┘
```

Parameters

ECHO
Requests (for 3270-type devices only) that the system to save the line of data being input in the current page (as displayed on the screen). If ECHO is not specified, data entered will not be retained and, therefore, will not be available for review by the terminal operator.

NOBACKPAGE
Requests (for 3270-type devices only) that the system not save previously input pages in a scratch area. If NOBACKPAGE is specified, the terminal operator can view only the current page of data. NOBACKPAGE is valid only with the first input request in a line mode session.

INTO <u>(input-data-location)</u>
Indicates the program variable-storage entry reserved for the input data. *Input-data-location* is the symbolic name of a user-defined field. The length of the data area is determined by one of the following specifications:

TO <u>(end-input-data-location)</u>
Indicates the end of program variable storage reserved for the input data stream and is specified following the last data-item entry in *input-data-location*. *End-input-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the data area reserved for the input data stream.

MAX LENGTH <u>(input-data-location-max-length)</u>
Defines the length, in bytes, of the input data stream. *Input-data-max-length* is either the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant.

If the input data stream is larger than the data area reserved in program variable storage, the system truncates the data to fit the available space.

RETURN LENGTH INTO <u>(input-data-actual-length)</u>
Indicates the location to which the system will return the actual length of the input data stream. *Input-data-actual-length* is the symbolic name of a user-defined field.

If the data stream has been truncated, *input-data-actual-length* contains the original length before truncation.

Example: The following statement reads the specified data from a 3270-type device into the specified location in the program and echoes the input data on the screen:

```
READ LINE FROM TERMINAL
   ECHO
   INTO (EMPL_DATA) TO (END_EMPL_DATA);
```

The following statement reads the specified data into the program without saving pages associated with the line I/O session:

```
READ LINE FROM TERMINAL
   NOBACKPAGE
   INTO (EMPL_DATA) MAX LENGTH (8)
   RETURN LENGTH INTO (REC_DATA_LENGTH);
```

Status codes: Upon completion of the READ LINE FROM TERMINAL function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4707	A logical or permanent I/O error has been encountered in the input data stream.
4719	The input area specified for the return of data is too small; the returned data has been truncated to fit the available space.
4731	The line request block (LRB) contains an invalid field, indicating a possible error in the program's parameters.
4732	The derived length of the specified line input area is zero or negative.
4738	The specified program variable-storage entry has not been allocated as required. A prior GET STORAGE request must be issued.
4743	The line I/O session has been canceled; the terminal operator has pressed CLEAR (3270s), ATTENTION (2741s), or BREAK (teletypes).

6.60 READ TERMINAL (DC/UCF)

The READ TERMINAL statement requests a synchronous or asynchronous basic mode data transfer from the terminal to program variable storage.

Syntax

```
►►── READ TERMINAL ─┬──────────────┬───────────────────────────────────────►
                    ├─ WAIT ← ─────┤
                    └─ NOWAIT ─────┘

►─┬──────────────────────────────────────────────────────┬────────────────►
  └─┬─ MODIFIED ─┬─ FROM POSITION (screen-position) ─────┘
    └─ BUFFER ───┘

►── INTO (input-data-location) ─┬─ TO (end-input-data-location) ──────────┬──►
                                └─ MAX LENGTH (input-data-max-length) ────┘

►─┬─────────────────────────────────────────────────┬─ ; ────────────────►◄
  └─ RETURN LENGTH INTO (input-data-actual-length) ─┘
```

Parameters

WAIT

Specifies that the read operation will be synchronous; the issuing task will automatically relinquish control to the system and must wait for completion of the read operation before processing can continue. WAIT is the default.

NOWAIT

Specifies that the read operation will be asynchronous; the issuing task will continue executing.

Note: If NOWAIT is specified, the program must issue a CHECK TERMINAL request (described later in this chapter) before performing any other I/O operations.

MODIFIED/BUFFER

Requests (for 3270-type devices only) that the system transfer data to the application program without requiring the terminal operator to signal completion of data entry.

MODIFIED

Reads all modified fields in the terminal buffer into variable storage.

BUFFER

Executes a READ BUFFER command that reads the entire contents of the terminal buffer into variable storage.

FROM POSITION <u>(screen-position)</u>

Defines the buffer address (screen position) at which the read will start. *Screen-position* is either the symbolic name of a user-defined FIXED BINARY(31) field or the address itself enclosed in single quotation marks.

INTO <u>(input-data-location)</u>

Specifies the data area reserved for the input data stream. This parameter is not specified for asynchronous requests that use the CHECK TERMINAL statement to allocate storage for the input buffer. *Input-data-location* is the symbolic name of a user-defined field.

If the input data stream is larger than the specified data area, the system truncates the data to fit the available space.

TO **(end-input-data-location)**
Indicates the end of the data area reserved for the input data stream and is specified following the last data-item entry in *input-data-location*. *End-input-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the data area reserved for the input data stream.

MAX LENGTH **(input-data-max-length)**
Defines the length, in bytes, of the data area reserved for the input data stream. *Input-data-max-length* is either the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant.

RETURN LENGTH INTO **(input-data-actual-length)**
Indicates the location to which the system will return the actual length of the input data stream. *Input-data-actual-length* is the symbolic name of a user-defined field. If the data stream has been truncated, *input-data-actual-length* contains the original length before truncation.

Example: The following statement illustrates a basic mode request to read data from the terminal to the specified location in variable storage:

```
READ TERMINAL
   WAIT
   INTO (TERM_LINE) TO (END_TERM_LINE);
```

Status codes: Upon completion of the READ TERMINAL function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
000	The request has been serviced successfully.
4519	The input area specified for the return of data to the issuing program is too small; the returned data has been truncated to fit the available space.
4527	A permanent I/O error has occurred during processing.
4528	The dial-up line for the terminal has been disconnected.
4531	The terminal request block (TRB) contains an invalid field, indicating a possible error in the program's parameters.
4532	The derived length of the specified input data area is zero or negative.

Status code	Meaning
4535	Storage for the input buffer cannot be acquired because the specified program variable-storage entry has been previously allocated; no I/O has been performed.
4539	The terminal device associated with the issuing task is out of service.

6.61 READY

The READY statement prepares a database area for access by DML functions and specifies that area's usage mode.

The DBA can specify default usage modes in the subschema. Run units that use such a subschema need not issue any READY statements; the areas are automatically readied in the predefined usage modes. However, if a run unit issues a READY statement for one area, it must issue READY statements for all areas that it will access.

PROTECTED and EXCLUSIVE options: The specified usage mode can be qualified with a PROTECTED option to prevent concurrent update or an EXCLUSIVE option to prevent concurrent use of areas by other run units executing under the CA-IDMS/DB central version. Each area can be readied in its own usage mode. Usage modes can be changed by executing a FINISH statement (see 6.37, "FINISH" on page 6-89), then starting a new run unit by issuing a BIND RUN_UNIT statement, the appropriate BIND RECORD statements, and a READY statement specifying the new usage mode.

Ready areas individually or together: When the run unit readies database areas, all areas can be readied with a single READY statement or each area to be accessed can be readied individually. All areas affected explicitly or implicitly by the DML statements issued by the run unit must be readied. Other areas included in the subschema need not be readied.

Position of READY statements: The READY statement can appear anywhere within an application program; however, to avoid runtime deadlock, the best practice is to ready all areas before issuing any other DML statements. A BIND RUN_UNIT statement must be processed successfully before a READY statement can be issued.

You can use the READY statement in both navigational and Logical Record Facility (LRF) environments.

Syntax:

```
►►── READY ─┬──────────────────────┬─┬──────────────────────────────────────┬─ ; ─►◄
            └─ AREA (area-name) ───┘ └─┬─────────────┬─┬─ RETRIEVAL ← ─┬──┘
                                       ├─ PROTECTED ─┤ └─ UPDATE ──────┘
                                       └─ EXCLUSIVE ─┘
```

Parameters

AREA (area-name)

Opens only the specified area. *Area-name* must be an area included in the subschema. If *area-name* is not specified, the READY statement opens all areas included in the subschema.

RETRIEVAL
Opens the area for retrieval only and allows other concurrently executing run units to open the same area in any usage mode other than one that is exclusive. RETRIEVAL is the default.

UPDATE
Opens the area for both retrieval and update and allows other concurrently executing run units to open the same area in any usage mode other than one that is exclusive or protected.

PROTECTED
Prevents concurrent update of the area by run units executing under the same central version. Once a run unit has readied an area with the PROTECTED option, no other run unit can ready that area in any UPDATE usage mode until the first run unit releases it by means of the FINISH statement (see 6.37, “FINISH” on page 6-89 earlier in this chapter). A run unit cannot ready an area with the PROTECTED option if another run unit has readied the area in UPDATE usage mode or with the EXCLUSIVE option.

If neither PROTECTED nor EXCLUSIVE is specified, the default usage mode of shared is invoked.

If a READY statement would result in a usage mode conflict for an area, while running under the CA-IDMS/DB central version, the run unit issuing the READY is placed in a wait state on the first functional database call.

EXCLUSIVE
Prevents concurrent use of the area by any other run unit executing under the CA-IDMS/DB central version. Once a run unit has readied an area with the EXCLUSIVE option, no other run unit can ready that area in any usage mode until the first run unit releases it.

If neither PROTECTED nor EXCLUSIVE is specified, the default usage mode of shared is invoked.

If a READY statement would result in a usage mode conflict for an area, while running under the CA-IDMS/DB central version, the run unit issuing the READY is placed in a wait state on the first functional database call.

Note: Modification statements involving areas opened in one of the update usage modes are not valid if they affect sets that include records in an area opened in one of the retrieval usage modes.

Example: The following statement readies all subschema areas in a usage mode of PROTECTED UPDATE:

```
READY PROTECTED UPDATE;
```

Status codes: Upon completion of the READY function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0910	The subschema specifies an access restriction that prohibits readying the area in the specified usage mode.
0923	The named area is not in the subschema.
0928	The run unit has attempted to ready an area that has been readied previously.
0966	The area is not available in the requested usage mode. If running in local mode, the area is locked against update. If running under the central version, either the area is offline to the central version, or an update usage mode was requested and the area is in retrieval mode to the central version.
0970	The database will not ready properly; a JCL error is the probable cause.
0971	The page group/page range for the area being readied could not be found in the DMCL.
0978	A READY has been issued after the first functional call; it is recommended that all areas be readied before the first functional call is issued.

6.62 RETURN (DC/UCF)

The RETURN statement retrieves the database key for an indexed record without retrieving the record itself, thus establishing currency in the indexed set. The record's symbolic key is moved into the data fields within the record in program variable storage. The contents of all non-key fields for the record are unpredictable after the execution of the RETURN verb. Optionally, the program can indicate that the symbolic key can be moved into some other specified variable storage location.

Current of index is established by:

- Successful execution of the RETURN statement, which sets current of index at the index entry from which the database key was retrieved.
- A status code of 1707 (end of index), which sets currency on the index owner. The DBMS returns the owner's database key.
- A status code of 1726 (index entry not found), which sets current of index as follows:
  - Between the two entries that are higher and lower than the specified value
  - After the highest entry, if the specified value is higher than all index entries
  - Before the lowest entry, if the specified value is lower than all index entries

You can use the RETURN statement in navigational and Logical Record Facility (LRF) environments.

Note: The DML precompiler views an incorrectly formatted RETURN statement as a PL/I RETURN function and does not flag the error. The incorrect RETURN DML statement is passed to the PL/I precompiler without expansion into a CALL statement, causing compile-time errors.

Syntax

```
►►── RETURN CURRENCY SET (index-set-name) ─┬─────────┬──────────────────────►
                                          ├─ FIRST ─┤
                                          ├─ LAST ──┤
                                          ├─ NEXT ──┤
                                          └─ PRIOR ─┘

►── INTO (db-key-field)─┬──────────────────────────────────┬─────── ; ──►◄
                        └─ KEY INTO (symbolic-key-field) ──┘
```

Parameters

RETURN CURRENCY SET (index-set-name)
Identifies the indexed set from which the specified database key is to be returned.

FIRST
Retrieves the database key for the first index entry.

LAST
Retrieves the database key for the last index entry.

NEXT
Retrieves the database key for the index entry following current of index. If the current of index is the last entry, a status code of 1707 (end of index) is returned.

PRIOR
Retrieves the database key for the index entry preceding current of index. If the current of index is the first entry, a status code of 1707 (end of index) is returned.

INTO (db-key)
Identifies the field to which the database key is returned. *Db-key* is the symbolic name of a user-defined FIXED BINARY(31) field.

KEY INTO (symbolic-key)
Saves the symbolic key (CALC, sort, or index) of the specified record. *Symbolic-key* is the name of a user-defined alphanumeric field into which the symbolic key of the specified record will be returned. *Symbolic-key* must be large enough to contain the largest contiguous or noncontiguous symbolic key.

If the KEY INTO clause is not specified, the key will be moved into the corresponding fields in the user record's storage.

Syntax

```
►►── RETURN USING (index-key-value) SET (index-set-name) ─────────────────►

►── INTO (db-key-field)─┬──────────────────────────────────┬──────── ; ─►◄
                        └─ KEY INTO (symbolic-key-field) ──┘
```

Parameters

RETURN USING (index-key-value)
Retrieves the database key for the first index entry whose symbolic key equals *index-key-value* (If no such entry exists, a status of 1726 (index entry not found) is returned.):

SET (index-set-name)
Identifies the indexed set from which the specified database key is to be returned.

INTO (db-key)
Identifies the field to which the database key is returned. *Db-key* is the symbolic name of a user-defined FIXED BINARY (31) field.

KEY INTO (symbolic-key)
Saves the symbolic key (CALC, sort, or index) of the specified record. *Symbolic-key* is the name of a user-defined alphanumeric field into which the symbolic key of the specified record will be returned. *Symbolic-key* must be large enough to contain the largest contiguous or noncontiguous symbolic key.

If the KEY INTO clause is not specified, the key will be moved into the corresponding fields in the user record's storage.

Example: The following RETURN statement retrieves the database key for the first index entry in the EMP_LNAME_NDX set and moves the record's symbolic key into the INT_INDEX_KEY field.

```
RETURN CURRENCY SET (EMP-LNAME-NDX)
   FIRST INTO (INT-INDEX-KEY);
```

Status codes: Upon completion of the RETURN function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
0057	A retrieval-only run unit has detected an inconsistency in an index that should cause an 1143 abend, but optional APAR bit 216 has been turned on.
1701	The area in which the object record or its index owner participates has not been readied.
1707	Either the end of the indexed set has been reached or the indexed set is empty.
1725	Currency has not been established for the specified indexed set.
1726	Record not found.
1763	The indexed set has not been registered with IDMSIXUD for the subschema in use.

6.63 ROLLBACK

The ROLLBACK statement requests recovery of the recovery unit (that part of a run unit that falls between two checkpoints) and, optionally, allows the run unit to continue database accessing activities without reissuing the necessary BIND and READY commands. Recovery restores database records, scratch areas, and queues to their condition at the most recent begin or commit checkpoint written for the issuing run unit or task.

Note: ROLLBACK TASK applies to CA-IDMS/DC only.

The way in which recovery is effected depends on whether the run unit is running under the CA-IDMS/DB central version or in local mode and whether the journal file is on disk or tape:

- **Recovery is effected automatically** if the run unit issuing the ROLLBACK statement is running under the central version and the central version is journaling to a disk file. The central version continues to process other applications during recovery. If the last checkpoint was not established by a COMMIT TASK or FINISH TASK function, the database, scratch, and queue areas may have checkpoints that do not occur at the same point in the journal file.

- **Recovery is not effected automatically** under the following circumstances:

  - If the run unit is running under the central version and the central version is journaling to a tape file

  - If the run unit is executing in local mode

  In these cases, the ROLLBACK statement causes the affected areas to be flagged for subsequent recovery by the ROLLBACK utility statement, which rolls back the database. If the journal file is on disk, you must archive the file to tape using the ARCHIVE JOURNAL utility statement before using the ROLLBACK utility statement.

  ►► For information on the ROLLBACK and ARCHIVE JOURNAL utility statements, refer to *CA-IDMS Utilities*.

You can use the ROLLBACK statement in both the navigational and the Logical Record Facility (LRF) environments.

Syntax

```
►►── ROLLBACK ─┬──────┬─┬────────────┬─ ; ──────────────────────────►◄
               └ TASK ┘ └ (CONTINUE) ┘
```

Parameters

TASK
Restores all run units associated with the issuing task to the most recent checkpoint. TASK only applies to CA-IDMS/DC.

CONTINUE
Rolls back the issuing run unit (ROLLBACK CONTINUE) or all run units associated with the issuing task (ROLLBACK TASK CONTINUE), but does not terminate the run unit. Database access can be resumed without reissuing BIND and READY statements. CONTINUE applies only to programs running under the CA-IDMS/DB central version.

Example: The following statement clears the system buffers, writes an abort checkpoint to the journal file, restores database records to the most recent checkpoint, nullifies all currencies, and terminates the run unit:

```
ROLLBACK;
```

Status codes: Upon completion of the ROLLBACK function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
5031	The specified request is invalid; the program may contain a logic error.
5096	Too many run units are currently active; check the system log for details.
5097	An invalid status has been received from DBIO/DBMS; check the system log for details.

6.64 SEND MESSAGE (DC/UCF)

The SEND MESSAGE statement sends a message to another terminal or user or to a group of terminals or users defined as a *destination* during system generation. The SEND MESSAGE function does not employ the data dictionary message area; instead, the system places each message in a queue, sending the message to the appropriate terminal only when it is possible to do so without disrupting executing tasks. Typically, the system sends queued messages to a terminal the next time the ENTER NEXT TASK CODE message is displayed.

Syntax

```
►►── SEND MESSAGE ─┬────────────┬─ TO ─┬─ DEST ID (destination-id) ─┬──────────►
                   ├─ ONLY ← ───┤      ├─ USER ID (user-id) ────────┤
                   └─ ALWAYS ───┘      └─ LTERM ID (lterm-id) ──────┘

 ►── FROM (message-location) ─┬─ TO (end-message-location) ─┬─ ; ──────────►◄
                              └─ LENGTH (message-length) ───┘
```

Parameters

ONLY/ALWAYS
Specifies whether the system is to queue the message if the specified destination, user, or terminal is not currently available:

ONLY
Sends the message immediately if the destination, user, or terminal is available, and not to queue the message for subsequent transmission if the destination, user, or terminal is not available.

Note: If ONLY is specified with the DEST ID option (described below) and if some, but not all, of a group of users or terminals in the destination are available, the system will send the message to those available. The sender will not be aware of any unsuccessful transmissions.

ALWAYS
Sends the message immediately if the destination, user, or terminal is available, and to queue the message for later transmission if the destination, user, or terminal is not available.

TO
Specifies the destination, user, or logical terminal to receive the message:

DEST ID (destination-id)
Identifies the recipient of the message as a destination. The specified destination must have been defined during system generation. *Destination-id* is either the symbolic name of a user-defined field that contains the destination ID or the ID itself enclosed in quotation marks.

USER ID (user-id)
Identifies the user to receive a message. The specified user can be signed on to any terminal. *User-id* is the symbolic name of a user-defined field that contains the user ID.

LTERM ID **<u>(lterm-id)</u>**
Identifies the logical terminal to receive the message. *Lterm-id* is either the symbolic name of a user-defined field that contains the terminal ID or the id itself enclosed in quotation marks.

FROM **<u>(message-location)</u>**
Specifies the program variable-storage entry that contains the text of the message to be sent. *Message-location* is the symbolic name of a user-defined field. The length of the message text is determined by one of the following specifications:

TO **<u>(end-message-location)</u>**
Indicates the end of the program variable-storage entry that contains the message text and is specified following the last field in *message-location*. *End-message-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the message text.

LENGTH **<u>(message-length)</u>**
Defines the length, in bytes, of the message text. *Message-length* is either the symbolic name of a user-defined field that contains the length or the length itself expressed as a numeric constant.

Examples: The following statement sends the message in the TERM_MESS field to the logical terminal KENNEDYA:

```
SEND MESSAGE ALWAYS
   TO LTERM ID ('KENNEDYA')
   FROM (TERM_MESS) TO (END_TERM_MESS);
```

The following statement sends the message in the TERM_MESS field to the user KYJOE2:

```
SEND MESSAGE
   TO USER ID ('KYJOE2')
   FROM (TERM_MESS) TO (END_TERM_MESS);
```

The following statement sends the message in the TERM_MESS field to the destination ALL:

```
SEND MESSAGE
   TO DEST ID ('ALL')
   FROM (TERM_MESS) TO (END_TERM_MESS);
```

Status codes: Upon completion of the SEND MESSAGE function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4907	An I/O error has occurred during processing.
4921	The specified message recipient has not been defined.

Status code	Meaning
4931	The parameter list is invalid.
4932	The derived length of the specified message data area is zero or negative.
4938	The specified program variable storage has not been allocated, as required. A GET STORAGE request must be issued.

6.65 SET TIMER (DC/UCF)

The SET TIMER statement defines an event that is to occur after a specified time interval or cancels the effect of a previously issued SET TIMER request. Using the SET TIMER function, a program can:

- Delay task processing for a specified period of time
- Post an ECB at the end of a specified period of time
- Initiate a task at the end of a specified period of time

Syntax

```
►►── SET TIMER ─┬─ WAIT ───┬───────────────────────────────────────────►
                ├─ POST ───┤
                ├─ START ──┤
                └─ CANCEL ─┘

►─┬─────────────────────────────────────────────────────┬──────────────►
  └─ INTERVAL (time-before-action-taken) SECONDS ─┘

►─┬──────────────────────┬─┬────────────────────────────────────────────►─
  └─ EVENT (post-ecb) ─┘ └─ TASK CODE (start-task-code) ────────────────

►───────────────────────────────────────────────────────────────────────►
 ────────────────────────┘
  └─ PRIORITY (priority) ─┘

►─┬──────────────────────────┬──────────────────────────────────────────►
  └─ TIMER ID (ice-address) ─┘

►───────────────────────────────────────────────────────────────────────►─
  └─ DATA FROM (start-task-data-location) ──────────────────────────────

►──────────────────────────────────────────────┬─ ; ────────────────────►◄
 ─┬─ TO (end-start-task-data-location) ──────┤
  └─ LENGTH (start-task-data-location) ──────┘
```

Parameters

WAIT/POST/START/CANCEL
Establishes a time-related event or cancels a previously requested time-dependent action.

WAIT
Places the issuing task in a wait state and instructs the system to redispatch the issuing task after the specified time interval elapses. Because WAIT relinquishes control until the time interval has elapsed, a subsequent SET TIMER request cannot be used to cancel this WAIT request.

POST
Posts a user-specified ECB after the specified time interval elapses; the issuing task continues to run. If POST is specified, the EVENT parameter (described below) must also be specified.

START
Initiates a user-specified task after the specified time interval elapses. If START is specified, the TASK CODE parameter (described below) must also be specified.

CANCEL
Cancels the effect of a previously issued SET TIMER request.

INTERVAL (time-before-action-taken) SECONDS
Specifies (for WAIT, POST, START requests only) the time in seconds from the issuance of a SET TIMER request at which the requested event will occur. *Time-before-action-taken* is either the symbolic name of a user-defined field that contains the time interval or the interval itself expressed as a numeric constant.

Note: For efficiency reasons, the time when the event is to occur is calculated by adding the *time-before-action-taken* value to the time at which the last TICKER interval expired. Therefore, the actual interval before the event occurs may vary plus or minus from *time-before-action-taken* by an amount up to the TICKER interval. Refer to the *CA-IDMS System Generation* manual for more information on the TICKER interval.

EVENT (post-ecb)
Specifies (for POST requests only) the ECB to be posted. *Post-ecb* is the symbolic name of a user-defined area composed of three binary fullword fields that contain the ECB.

TASK CODE (start-task-code)
Specifies (for START requests only) the 1- to 8-character code of the task to be initiated. *Start-task-code* is either the symbolic name of the user-defined field that contains the task code or the task code itself enclosed in quotation marks. The specified task code must have been defined to the system during system generation or at run time with a DCMT VARY DYNAMIC TASK command.

PRIORITY (priority)
Specifies a dispatching priority for the task. *Priority* is either the symbolic name of a user-defined field that contains the priority or the priority itself expressed as a numeric constant in the range 0 through 240. The new task's priority defaults to the priority defined for that task code.

TIMER ID (ice-address)
Specifies (for POST, START, CANCEL requests only) the address of the interval control element (ICE) associated with the timed event. *Ice-address* is the symbolic name of a user-defined FIXED BINARY(31) field. If either POST or START has been specified, *ice-address* references a field to which the system will return the ICE address. If CANCEL has been specified, *ice-address* references the field that contains the ICE address returned by the system following a SET TIMER POST or SET TIMER START request.

Note: The TIMER ID parameter must be specified with SET TIMER POST and SET TIMER START requests if the program is to issue subsequent SET TIMER CANCEL requests.

DATA FROM (start-task-data-location)
Specifies (for START requests only) the user data to be passed to the new task. *Start-task-data-location* is the symbolic name of a user-defined field that contains the data to be passed. The length of the data area is determined by one of the following specifications:

TO (end-start-task-data-location)
Indicates the end of the data area being passed to the new task and is specified following the last data-item entry in *start-task-data-location*. *End-start-task-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the data area being passed.

LENGTH (start-task-data-location)
Specifies the length, in bytes, of the data area. *Start-task-data-location* is either the symbolic name of a user-defined program variable storage field that contains the length of the data area or the length itself expressed as a numeric constant.

Note: When the new task is started, the first program which receives control in the new task can access this data by observing the following conventions:

- The receiving program must access the data as though it had been passed by an Assembler program.
- The data will be preceded by a half-word field containing the length of the original data.

Examples: The following statement places the issuing task in a wait state and redispatches it after nine seconds have elapsed:

```
SET TIMER WAIT
   INTERVAL (9) SECONDS;
```

The following statement posts the event PODB after five seconds have elapsed:

```
SET TIMER POST
   INTERVAL (5) SECONDS
   EVENT ('PODB')
   TIMER ID (TMR_ID);
```

The following code declares a data field, starts the SPSG task after five seconds have elapsed, and passes the specified data to the task:

```
DECLARE 1 PASSED_DATA,
          2 PASSED_FIXED FIXED,
          2 PASSED_CHAR CHAR(20),
          2 PASSED_END CHAR(1);
SET TIMER START
   INTERVAL (5) SECONDS
   TASK CODE ('SPSG')
   DATA FROM (PASSED_DATA) TO (PASSED_END);
```

The following code in the program invoked by task SPSG establishes access to the data passed by the above SET TIMER START command:

```
SPSGPRG: PROC (PARMIN_DUMMY)
   OPTIONS(MAIN,REENTRANT) REORDER;

DECLARE 1 PARMIN_DUMMY FIXED;
DECLARE 1 PARMIN BASED (ADDR(PARMIN_DUMMY)),
          2 PASSED_DATA_LENGTH FIXED BIN(15),
          2 PASSED_DATA,
            3 PASSED_FIXED FIXED,
            3 PASSED_CHAR CHAR(20);
```

The following statement cancels the timed event referenced by TMR-ID:

```
SET TIMER CANCEL
   TIMER ID (TMR_ID);
```

Status codes: Upon completion of the SET TIMER function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
3516	The interval control element (ICE) specified for a SET TIMER CANCEL request cannot be found.
3532	The derived length of the data area is negative.

6.66 SNAP (DC/UCF)

The SNAP statement requests a memory snap of one or all of the following areas:

- **Task areas** — Includes all resources associated with the issuing task, as well as the task control element (TCE) and dispatch control element (DCE) for the task. Information displayed by the snap is formatted with headers.
- **System areas** — Includes areas for all tasks and internal system control blocks. Task areas are not itemized separately. Information displayed by the snap is formatted with headers.
- **Specified locations in memory** — Includes one or more areas of memory specifically requested by location and length. The information displayed is not formatted with headers.

The areas requested in the SNAP request are written to the system log file, which is defined during system generation as a sequential dataset or a dictionary area.

Syntax

```
►►── SNAP ─┬──────────────────┬──────────────────────────────────────────►
           └─ TITLE (title) ──┘

 ►─┬─────────────────────────────────────────────────────────────┬─; ─►◄
   ↓                                                             │
   ├─┬─ ALL ─────────────────────────────────────────────────┬───┘
     ├─ SYSTEM ──────────────────────────────────────────────┤
     ├─ TASK ────────────────────────────────────────────────┤
     │ ┌──────────────────────────────────────────────────┐  │
     └─↓── FROM (begin-snap-location) ─┬─ TO (end-snap-location) ─┬─┴──┘
                                       └─ LENGTH (snap-length) ───┘
```

Parameters

TITLE <u>(title)</u>
Specifies the title to be printed at the beginning of each page of the snap. If requested, a title must contain 134 characters; the first character is reserved for use by the system, and the second character must be a valid ASA carriage control character (blank, 0, 1, +, or -). *Title* is the symbolic name of a user-defined field that contains the title.

ALL/SYSTEM/TASK
Requests a formatted snap of specified areas.

ALL
Writes a snap of both task and system areas. Areas associated with the issuing task are formatted separately from the system areas. (Task areas are also included with the system areas but are not itemized by task.)

SYSTEM
Writes a snap of system areas.

TASK
Writes a snap of task areas.

FROM <u>(begin-snap-location)</u>
Writes a snap of the specified memory location. *Begin-snap-location* is the symbolic name of a user-defined field that indicates the starting location of the area to be snapped.

TO <u>(end-snap-location)</u>
Indicates the end of the area to be snapped and is specified following the last data-item to be included in the snap. *End-snap-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the area requested in the snap.

LENGTH <u>(snap-length)</u>
Defines the length, in bytes, of the area to be included in the snap. *Snap-length* is either the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant.

Example: The following example illustrates a SNAP statement that writes a memory snap of the specified memory location:

```
SNAP TITLE (SNAP_TITLE)
   FROM (START_LOC) TO (END_LOC);
```

Status codes: Upon completion of the SNAP function, the ERROR_STATUS field in the system communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4032	The derived length of the specified snap storage area is zero or negative.

6.67 STARTPAGE (DC/UCF)

The STARTPAGE statement initiates a paging session. It can be followed by any number of DML commands, including MAP IN and MAP OUT commands. The map paging session is terminated by an ENDPAGE command (or by another STARTPAGE command, if one is encountered before an ENDPAGE command).

Note: Only *one* pageable map can be handled by the statements enclosed by a given STARTPAGE/ENDPAGE pair.

Syntax

```
►►── STARTPAGE session (map-name) ─────────────────────────────────────►

►─┬──────────┬─┬──────────────┬─┬────────────┬─┬──────────────────┬─ ; ──►◄
  ├─ WAIT ───┤ ├─ BACKPAGE ← ─┤ ├─ UPDATE ← ─┤ ├─ AUTODISPLAY ← ─┤
  ├─ NOWAIT ← ┤ └─ NOBACKPAGE ─┘ └─ BROWSE ───┘ └─ NOAUTODISPLAY ─┘
  └─ RETURN ─┘
```

Parameters

map-name
Specifies the 1- to 8-character name of a map specified by the DECLARE MAP statement, as described in Chapter 5, "DML Precompiler-Directive Statements" on page 5-1. The STARTPAGE command must precede any commands (such as MAP IN) that specify operations to be performed using the map.

WAIT/NOWAIT/RETURN
Specifies the runtime flow of control when the operator presses a control key.

WAIT
Specifies that runtime mapping automatically handles paging transactions that do not cause data to be updated. Control is passed to the program when the terminal operator presses a control key that requests an update or nonpaging operation.

NOWAIT
Specifies that runtime mapping automatically handles all paging and update transactions. Control is passed to the program only when neither an update nor a paging request is made when the operator presses a control key. NOWAIT is the default.

RETURN
Specifies that runtime mapping does not handle any terminal transactions in the paging session. Control is passed to the program whenever the operator presses a control key.

Runtime mapping does not update program variable storage unless a MAP IN command is issued. In cases where the operator can update data, it is recommended that WAIT or RETURN be specified for the session so that data can be retrieved as it is updated.

BACKPAGE/NOBACKPAGE
Specifies whether the terminal operator can display a previous map page.

BACKPAGE
Specifies that the operator can display previous pages of detail occurrences. BACKPAGE is the default.

NOBACKPAGE
Specifies that the operator cannot display any page of detail occurrences with a page number lower than the current page number. Modifications made on a given page of the map must be requested by MAP IN statements in the application program before a MAP OUT RESUME command is issued. The previous page of detail occurrences is deleted from the session scratch record when a new map page is displayed.

Note: NOBACKPAGE cannot be assigned if UPDATE and NOWAIT are specified for the session.

UPDATE/BROWSE
Specifies whether the terminal operator can modify map data fields.

UPDATE
Specifies that the terminal operator can modify variable map fields, subject to restrictions specified for the map either at map definition time or by statements in the program. UPDATE is the default.

BROWSE
Specifies that the terminal operator can modify only the page field (if any) of the map. The MDTs for variable fields on the map can be set on only according to specifications made either in the map definition or by statements in the program.

AUTODISPLAY/NOAUTODISPLAY
Specifies whether to override the automatic mapout that occurs when the first page of a map is built.

AUTODISPLAY
Enables automatic display of the pageable map's first page. AUTODISPLAY is the default.

NOAUTODISPLAY
Disables automatic display of the pageable map's first page. You display the first page manually by using a MAP OUT RESUME statement.

Example: The following statement initiates a paging session in which the operator can page forward and backward within the pageable map but can make no modifications:

```
STARTPAGE SESSION (EMPMAPPG)
   NOWAIT BACKPAGE BROWSE;
```

Status codes: Upon completion of the STARTPAGE function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4604	A paging session was already in progress when this STARTPAGE command was received. An implied ENDPAGE was processed before this STARTPAGE was successfully executed.

6.68 STORE RECORD

The STORE RECORD statement performs the following functions:

- Acquires space and a database key for a new record occurrence in the database
- Transfers the value of the appropriate elements from program variable storage to the specified record occurrence in the database
- Connects the new record occurrence to all sets for which it is defined as an automatic member

Steps before executing STORE RECORD: Before executing the STORE RECORD statement, satisfy the following conditions:

- Ready all areas affected either implicitly or explicitly in one of the update usage modes (see 6.61, "READY" on page 6-164, earlier in this chapter).
- Make sure the program initializes all control elements (that is, CALC and sorted set control fields).
- If the record being stored has a location mode of DIRECT, initialize the contents of DIRECT_DBKEY (positions 197-200 of the IDMS communications block, as described in Chapter 3, "Communications Blocks and Error Detection" on page 3-1) with a suggested db-key value or a null db-key value of -1.
- If the record is to be stored in a native VSAM relative-record data set (RRDS), initialize the contents of DIRECT_DBKEY with the relative-record number that represents the location within the data set where the record is to be stored.
- Include in the subschema all sets in which the named record is defined as an automatic member, and the owner record of each of those sets. Sets for which the named record is defined as a manual member need not be defined in the subschema since the STORE RECORD statement does not access those sets. (An automatic member is connected automatically to the selected set occurrence when the record is stored; a manual member is not connected automatically to the selected set occurrence.)
- If the record being stored has a location mode of VIA, establish currency for that VIA set, regardless of whether the record being stored is an automatic or manual member of that set. Current of the VIA set provides the suggested page for the record being stored.
- Establish currency for all set *occurrences* in which the stored record will participate as an *automatic* member. Depending on set order, the STORE RECORD statement uses currency as follows:
  - If the named record is defined as a member of a set that is ordered FIRST or LAST, the record that is current of set establishes the set occurrence to which the new record will be connected.
  - If the named record is defined as a member of a set that is ordered NEXT or PRIOR, the record that is current of set establishes the set occurrence into

which the new record will be connected *and* determines its position within the set.

– If the named record is defined as a member of a sorted set, the record that is current of set establishes the set occurrence into which the new record will be connected. The DBMS compares the sort key of the new record with the sort key of the current record of set to determine if the new record can be inserted into the set by movement in the next direction. If it can, the current of set remains positioned at the record that is current of set and the new record is inserted. If it cannot, the DBMS finds the owner of the current of set (not necessarily the current occurrence of the owner record type) and moves as far forward in the next direction as is necessary to determine the logical insertion point for the new record.

Location modes: A record is stored in the database based on the location mode specified in the schema definition of the record. The location modes are as follows:

- **CALC** — The record being stored is placed on or near a page calculated by IDMS-DB from a control element (the CALC key) in the record.
- **VIA** — The record being stored is placed either as close as possible to the current of set (if current of set and member record occurrences share a common page range) or in the same relative position in the member record's page range as the current of set is in its associated page range (if current of set and member record occurrences do not share a common page range).
- **DIRECT** — The record being stored is placed on or near a user-specified page as determined by the value in the DIRECT_DBKEY field of the IDMS-DB communications block. If DIRECT_DBKEY contains a valid db-key for the record being stored, the DBMS assigns a db-key on the same page if space is available to the new record occurrence. Otherwise, it assigns the next available db-key, subject to the page-range limits of the record being stored. If DIRECT_DBKEY contains a value of -1, the first db-key available in the page range in which the record is to be stored is assigned to the record. In any case, the db-key of the stored record occurrence is returned to DBKEY (positions 13-16 in the CA-IDMS/DB communications block). The contents of DIRECT_DBKEY remain unchanged.

Currency: Following successful execution of a STORE RECORD statement, the stored record becomes current of run unit, its record type, its area, and all sets in which it participates as owner or automatic member.

Syntax

```
►►── STORE RECORD (record-name); ──────────────────────────────►◄
```

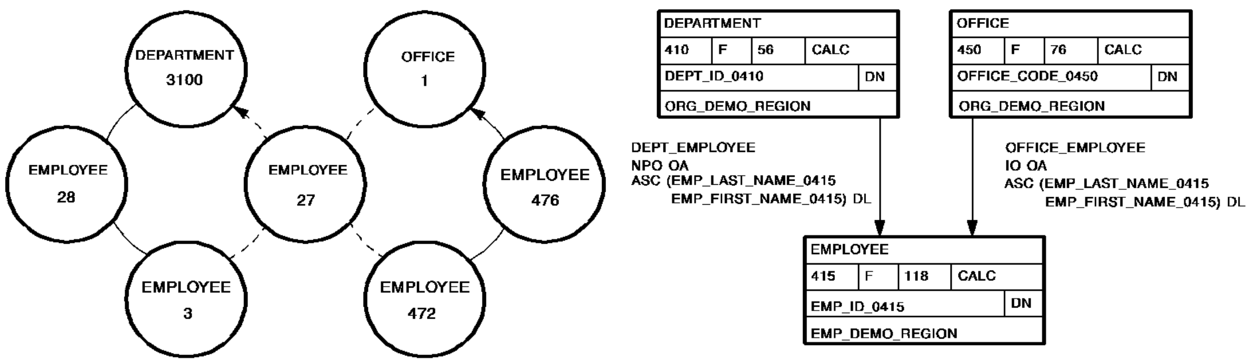
Parameter

record-name
Defines the named record occurrence, as specified in program variable storage. *Record-name* must specify a record type included in the subschema.

The ordering rules for each set govern the insertion point of the specified record in the set.

Example: The following figure illustrates the steps necessary to add a new EMPLOYEE record to the database. Since EMPLOYEE is defined as an automatic member of both the DEPT_EMPLOYEE and OFFICE_EMPLOYEE sets, currency must be established in each of those sets before issuing the STORE RECORD.

The first two DML statements establish OFFICE 1 and DEPARTMENT as current of the OFFICE_EMPLOYEE and DEPT_EMPLOYEE sets, respectively. When EMPLOYEE 27 is stored, it is connected automatically to each set.



	CURRENCIES RUN UNIT, RECORD, SET, AREA							
	RUN UNIT	DEPARTMENT	EMPLOYEE	OFFICE	DEPT_EMPLOYEE	OFFICE_EMPLOYEE	ORG_DEMO_REGION	EMP_DEMO_REGION
OFFICE_CODE = OFFICE_CODE_IN; FIND CALC RECORD (OFFICE);	1			1		1	1	
DEPT_ID = DEPT_ID_IN; FIND CALC RECORD (DEPARTMENT);	3100	3100		1	3100	1	3100	
STORE RECORD (EMPLOYEE);	27	3100	27	1	27	27	3100	27

Status codes: Upon completion of the STORE RECORD function, the ERROR_STATUS field in the IDMS-DB communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
1201	The area in which the named record is to be stored has not been readied.
1202	The suggested DIRECT_DBKEY value is not within the page range for the named record.
1205	Storage of the record would violate a duplicates-not-allowed option for a CALC record, a sorted set, or an index set.
1208	The named record is not in the subschema. The program has probably invoked the wrong subschema.
1209	The named record's area has not been readied in one of the update usage modes.
1210	The subschema specifies an access restriction that prohibits storage of the named record.
1211	The record cannot be stored in the area because of insufficient space.
1212	The record cannot be stored because no db-key is available. This is a system internal error.
1218	The record has not been bound.
1221	An area other than the area of the named record occurrence has been readied with an incorrect usage mode.
1225	A set occurrence has not been established for each set in which the named record is to be stored.
1233	At least one set in which the record participates as an automatic member has not been included in the subschema.
1253	The subschema definition of an indexed set does not match the indexed set's physical structure in the database.
1254	Either the prefix length of an SR51 record is less than zero or the data length is less than or equal to zero.
1255	An invalid length has been defined for a variable length record.
1260	A record occurrence that was encountered in the process of connecting automatic sets is inconsistent with the set named in the ERROR_SET field of the CA-IDMS/DB communications block; probable causes include a broken chain or improper database description.
1261	The record cannot be stored because of broken chains in the database.

6.69 STORE RECORD (LRF)

The STORE RECORD statement updates the database with field values for a logical-record occurrence. STORE RECORD does not necessarily result in storing new occurrences of all or any of the database records that participate in the logical record; the path selected to service a STORE RECORD logical-record request performs whatever database access operations the DBA has specified to service the request. For example, if an existing department gets a new employee, only the new employee information will be stored in the database; the department information need not be stored in the database because it already exists.

LRF uses field values present in the variable-storage location reserved for the logical record to make the appropriate updates to the database. You can optionally name an alternative storage location from which the new field values are to be obtained to perform the requested store operation.

Syntax

```
►►─── STORE RECORD (logical-record-location) ──────────────────────►

►─┬────────────────────────────────────┬─┬──────────────────────────┬─►
  └─ FROM (alt-logical-record-location) ─┘ └─ WHERE (boolean-expression) ─┘

►─┬──────────────────────────────────────────────┬─ ; ──────────►◄
  └─ ON LR_STATUS (path-status)  imperative-statement ─┘
```

Parameters

<u>logical-record-name</u>

Names the logical record to be stored. Unless the FROM clause (see below) is included, LRF uses field values present in the variable-storage location reserved for the specified logical record to make the appropriate updates to the database. *Logical-record-name* must specify a logical record defined in the subschema.

FROM <u>(alt-logical-record-location)</u>

Names an alternative variable storage location that contains the field values to be used to make appropriate updates to the database. When storing a logical record that has previously been retrieved into an alternative variable storage location, use the FROM clause to name the same area specified in the OBTAIN request. If the FROM clause is included in the STORE RECORD statement, *alt-logical-record-location* must identify a record location defined in program variable storage.

WHERE <u>(boolean expression)</u>

Specifies selection criteria to be applied to the object logical record.

►► For details on coding the WHERE clause, see 6.78, “Logical-record clauses (WHERE and ON)” on page 6-215 at the end of this chapter.

ON LR_STATUS <u>(path-status) imperative-statement</u>

Specifies the action to be taken if *path-status* is returned to the LR_STATUS field in the LRC block. *Path-status* must be a 1- to 16-character alphanumeric value.

►► For details on coding this clause, see 6.78, “Logical-record clauses (WHERE and ON)” on page 6-215 at the end of this chapter.

Example: The following example illustrates the steps necessary to store a new logical record, EMP-INSURANCE-LR, for a given employee:

```
EMP_ID_0415 = EMP_ID_IN;
INS_PLAN_CODE_0435 = INS_PLAN_IN;
SELECTION_DATE_0400 = S_DATE_IN;
TERMINATION_DATE_0400 = T_DATE_IN;
TYPE_0400 = TYPE_IN;
INS_PLAN_CODE_0400 = PLAN_IN;
STORE RECORD (EMP_INSURANCE_LR);
```

The following figure illustrates the new occurrence of the record EMP_INSURANCE_LR. The new occurrence of EMP_INSURANCE_LR consists of EMPLOYEE 149, INS_PLAN 001, and COVERAGE 'D'. The COVERAGE occurrence represents the only data physically added to the database.

	EMPLOYEE	INS-PLAN	COVERAGE
	149	002	M
	149	002	F
ONE OCCURRENCE OF EMP-INS-LR	149	001	D

6.70 TRANSFER (DC/UCF)

The TRANSFER statement is used to:

- Establish linkage with a specified program and to pass control and an optional parameter list to that program. The program issuing the TRANSFER RETURN request expects return of control at the instruction immediately following the TRANSFER statement when the linked program terminates or issues a DC RETURN request.
- Transfer control and an optional parameter list to a specified program. The program issuing the TRANSFER NORETURN request does *not* expect return of control.

Passing parameters from a non-PL/I program: If parameters are passed to a PL/I program from a non-PL/I program (CA-ADS, COBOL, Assembler), special code must be used in the PL/I program. A partial sample of this code is shown below:

```
SAMPPROC: PROCEDURE (F1,F2,F3) OPTIONS (MAIN,REENTRANT);
DCL  (F1,F2,F3) POINTER;
DCL  (SAMPSUBS SUBSCHEMA, SAMPSCHM SCHEMA) MODE (IDMS_DC) DEBUG;
DCL  IDMS ENTRY OPTIONS (INTER,ASM);
DCL  IDMSP ENTRY;
DCL  PASSED_FIELD_1 FIXED BIN (31) BASED(ADDR(F1));
INCLUDE IDMS (SUBSCHEMA_CTRL BASED(ADDR(F2)));
INCLUDE IDMS (RECORD_AA BASED(ADDR(F3)));
 .
 .
 .
rest of code
```

Here, a non-PL/I program has transferred control to this sample program, passing three parameters. The first is binary fullword. The second is the address of the subschema control block that the program will use. The third is a CA-IDMS/DB record. Note that dummy parameters are set up to provide addresses on which to base the structures that are actually passed.

Refer to the PL/I programmer's reference for your site for more information on passing parameters to a PL/I program from an Assembler program.

Note: The section (in the same reference) on invoking PL/I programs from COBOL programs is not relevant. In a DC/UCF environment, you must code the PL/I program as shown in the previous sample.

Syntax

```
►►── TRANSFER TO (program-name) ─┬──────────────┬─┬──────────────────────────┬─ ;─►◄
                                 ├─ RETURN ─────┤ │        ┌────── , ─────┐  │
                                 ├─ LINK ───────┤ └─ ( ─┴─ parameter ─┴─ ) ─┘
                                 ├─ NORETURN ← ─┤
                                 └─ XCTL ───────┘
```

Parameters

TO <u>(program-name)</u>

Specifies the 1- to 8-character name of the program to which control is transferred. *Program-name* is either the symbolic name of a user-defined field that contains the program name, or the name itself enclosed in quotation marks.

RETURN/NORETURN

Specifies whether control will be returned to the the calling program.

RETURN

Establishes linkage with the specified program, expecting return of control. The keywords RETURN and LINK are synonymous.

NORETURN

Transfers control to the specified program, not expecting return of control. The keywords NORETURN and XCTL are synonymous. NORETURN is the default.

<u>parameter</u>

Passes one or more parameters (data items) to the program receiving control. *Parameter* is the symbolic name of a user-defined field that contains the names of the data items to be passed. Multiple parameter specifications must be separated with a blank.

To use *parameter,* the DECLARE IDMSP ENTRY statement is required. For details on this PL/I declarative, see Chapter 4, “Required PL/I Declaratives” on page 4-1.

If *parameter* is specified, the data items being passed are defined in program variable storage for both the calling program and the linked program. The program receiving control must include a corresponding *parameter* clause in its PROCEDURE statement.

Examples: The following statement transfers control to the program in the PROGRAM_NAME field; the issuing program expects return of control:

```
TRANSFER TO (PROGRAM_NAME)
   LINK;
```

The following statement transfers control to PROGRAMD and passes three data items (FIELD_1, FIELD_2, and FIELD_3) to the program; the issuing program does not expect return of control:

```
TRANSFER TO ('PROGRAMD')
   NORETURN
   (FIELD_1, FIELD_2, FIELD_3);
```

Status codes: Upon completion of the TRANSFER function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.

Status code	Meaning
3020	The request cannot be serviced because an I/O, program-not-found, or potential deadlock error has occurred.

6.71 WAIT (DC/UCF)

The WAIT statement relinquishes control either to the system, pending completion of one or more events, or to a higher priority ready-to-run task. If control is relinquished to wait for the completion of one or more events, an event control block (ECB) must be defined for each event. If an ECB is already posted when the WAIT is issued, the task is redispatched immediately and control does not pass to another task.

Syntax

```
                              ┌──────────────────┐
►►── WAIT ──┬─┬─ LONG ← ─┬─┬─↓─ EVENT (ecb) ───┴──┬─┬───────────────────►
            │ └─ SHORT ──┘ └─ EVENT NAME (ecb-id) ─┘ │
            └─ REDISPATCH ───────────────────────────┘

►─┬──────────────────────────────────┬─ ; ──────────────────────────────►◄
  └─ STALL INTERVAL (stall-interval) ┘
```

Parameters

LONG/SHORT
Specifies whether the wait is expected to be of long-term or short-term duration.

LONG
Specifies that the wait is expected to be long-term. LONG should be specified for all waits expected to last a second or more (for example, terminal input). LONG is the default.

SHORT
Specifies that the wait is expected to be short-term. SHORT should be specified for all waits expected to last less than a second (for example, a disk I/O).

EVENT/EVENT NAME
Specifies an event upon which the issuing task is to wait.

EVENT (ecb)
Defines one or more ECBs upon which the task will wait. *ecb* is the symbolic name of a user-defined area that contains three binary fullword fields that contain the ECB. Multiple EVENT parameters must be separated by at least one blank.

EVENT NAME (ecb-id)
Specifies the 4-character symbolic ID of the ECB upon which the task will wait. *Ecb-id* is either the symbolic name of a user-defined field that contains the ECB ID, or the ID itself enclosed in quotation marks. Multiple EVENT NAME parameters cannot be specified.

REDISPATCH
Specifies that the issuing task wishes to relinquish control to any higher priority ready-to-run task before being redispatched.

STALL INTERVAL **(stall-interval)**
Indicates the time, in wall-clock seconds, that the system is to suspend processing of the issuing task. *Stall-interval* is the symbolic name of a user-defined fixed binary field containing the stall interval, or the interval itself expressed as a numeric constant.

Example: The following statement requests a short-term wait on the event PODB:

```
WAIT
   SHORT
   EVENT NAME ('PODB');
```

Status codes: Upon completion of the WAIT function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
3101	To wait on the specified ECB would cause a deadlock.

6.72 WRITE JOURNAL (DC/UCF)

The WRITE JOURNAL statement writes a task-defined record to the journal file. Records written to the journal file with the WRITE JOURNAL function will be available to user-defined exit routines during a task- or system-initiated rollback.

Syntax

```
►►── WRITE JOURNAL ─┬──────────────┬─┬────────────┬──────────────────────────►
                    ├─ WAIT ───────┤ ├─ SPAN ← ───┤
                    └─ NOWAIT ← ───┘ └─ NOSPAN ───┘

►── FROM (record-location) ─┬─ TO (end-record-location) ─┬─ ; ────────────►◄
                            └─ LENGTH (record-length) ───┘
```

Parameters

WAIT/NOWAIT
Specifies whether the issuing task is to wait for completion of the WRITE JOURNAL function before resuming execution:

WAIT
Specifies that the issuing task will wait for completion of the physical I/O associated with the WRITE JOURNAL function before resuming execution. This option will cause the system to write a partially filled buffer to the journal file.

NOWAIT
Specifies that the issuing task will not wait for completion of the WRITE JOURNAL function; the journal record will remain in a storage buffer until a future request necessitates writing the buffer to the journal file. NOWAIT is the default.

SPAN
Indicates that the system will write the record across several journal file blocks, if necessary. SPAN is the default.

Note: In general, the SPAN option provides better space utilization in the journal file than NOSPAN because it increases the average fullness of each block.

NOSPAN
Indicates that the system will write the record to a single journal file block; if it is longer than the journal block, the record will be split.

When a record is shorter than a journal file block, based on space available in the current journal block, the system will either place the record in the block, split it across multiple blocks (SPAN), or write it to a new block after the current block is written (NOSPAN).

The following considerations apply to using an exit routine to retrieve journal file records during recovery:

- If a WRITE JOURNAL statement issued before a failure specified the SPAN option, records may have been written across several journal blocks. To retrieve these records, the exit routine will be invoked once for each segment of each record to be retrieved.

- If a WRITE JOURNAL statement issued before a failure specified the NOSPAN option and records written to the journal file are shorter than journal blocks, the exit routine need only be concerned with the complete records.

FROM (record-location)
Defines the program variable-storage entry of the record to be written to the journal file. *Record-location* is the symbolic name of a user-defined field. The length of the record area is determined by one of the following specifications:

TO (end-record-location)
Indicates the end of the record area to be written to the journal file and is specified following the last data-item entry in *record-location*. *End-record-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the record being written to the journal file.

LENGTH (record-length)
Defines the length, in bytes, of the record to be written to the journal file. *Record-length* is either the symbolic name of the user-defined field that contains the length, or the length itself expressed as a numeric constant.

Example: The following statement writes the JOURNAL_DATA record to the journal file, spanning it across several blocks if necessary:

```
WRITE JOURNAL SPAN
   FROM (JOURNAL_DATA) TO (END_JOURNAL_DATA);
```

Status codes: Upon completion of the WRITE JOURNAL function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
5002	Storage is not available for the required control blocks.
5032	The derived length of the specified journal record is zero or negative.
5097	An invalid status has been received from DBIO/DBMS; check the system log for details.

6.73 WRITE LINE TO TERMINAL (DC/UCF)

The WRITE LINE TO TERMINAL statement transfers data from program variable storage to a terminal. WRITE LINE TO TERMINAL also establishes, modifies, and deletes page header lines.

Data transfers requested by WRITE LINE TO TERMINAL statements can be synchronous or asynchronous:

- **Synchronous** — After a synchronous request, control passes to the system. The system places the issuing task in an inactive state. For non-3270 devices, control does not return to the issuing program until the WRITE LINE TO TERMINAL request is complete. For 3270-type devices, all lines of output are saved in a buffer; the buffer is not transmitted to the terminal until it is full.

  The transfer of a line to the buffer will result in a processing delay; however, control returns to the program immediately following the request. If the line of data fills the buffer, the entire page of data must be transmitted to the terminal. In this case, control does not return to the issuing program until the terminal operator responds by pressing ENTER. Thus, the program is made conversational.

- **Asynchronous** — After an asynchronous request, control returns immediately to the issuing program. Thereafter, each time the program issues a line mode I/O request, the system automatically checks to determine if the last asynchronous request has completed and, therefore, whether a new data transfer can be initiated.

  With asynchronous requests, programs can buffer all required pages of output without suspending task execution during the actual transmission of data. However, the task can optionally terminate itself, thereby freeing resources and allowing the terminal operator to review the buffered output.

The system processes I/O requests in the sequence received from the task; thus, if a program issues a synchronous WRITE LINE TO TERMINAL request after issuing one or more asynchronous requests, the system will complete all I/O requests before returning control to the issuing program.

The WRITE LINE TO TERMINAL request issued automatically by the system to empty partially filled buffers upon completion of a task is synchronous; therefore, the terminal operator can view all screens and catch up with processing at that time. If an application allows the terminal operator to interrupt or terminate processing at some point within a task, a synchronous WRITE LINE TO TERMINAL request must be issued to suspend processing while awaiting an operator response.

Syntax

```
►►── WRITE LINE TO TERMINAL ─┬─────────┬─┬──────────┬─┬───────────────┬──────►
                             ├─ WAIT ← ┤ ├─ NEWPAGE ┤ └─ NOBACKPAGE ──┘
                             └─ NOWAIT ┘ └─ ERASE ──┘

►── FROM (output-data-location) ─┬─ TO (end-output-data-location) ─┬──────►
                                 └─ LENGTH (output-data-length) ───┘

►─┬──────────────────────────────┬─ ; ─────────────────────────────────►◄
  └─ HEADER (header-line-number) ┘
```

Parameters

WAIT
Specifies that the write operation is synchronous; the issuing task automatically relinquishes control and must wait for completion of the output operation before processing can continue. WAIT is the default.

NOWAIT
Specifies that the write operation is asynchronous; the issuing task continues executing.

NEWPAGE
Writes the output data line beginning on a new page. For 3270-type devices, the NEWPAGE option forces the system to output the contents of the current buffer, even if the buffer is not full. The keywords NEWPAGE and ERASE are synonymous.

NOBACKPAGE
Specifies (for 3270-type devices only) that pages output in a scratch area are not to be kept. If NOBACKPAGE is specified, the terminal operator can view only the current page of output. NOBACKPAGE is valid only with the first I/O request in a line mode session.

FROM (output-data-location)
Identifies the program variable-storage entry of the data to be transferred to the terminal device, or the page-header line being created, modified, or deleted. *Output-data-location* is the symbolic name of a user-defined field. The length of the output data stream is determined by one of the following specifications:

TO (end-output-data-location)
Indicates the end of the program variable-storage entry that contains the output data stream and is specified following the last data-item entry in *output-data-location*. *End-output-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the output data.

LENGTH (output-data-length)
Defines the length, in bytes, of the output data area. *Output-data-length* is either the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant.

Note: If the WRITE LINE TO TERMINAL statement is being used to delete a page-header line, *output-data-length* must be zero.

HEADER <u>(header-line-number)</u>
Specifies the number of the page header line being created, modified, or deleted. *Header-line-number* is either the symbolic name of a user-defined field that contains the header line number, or the header line number itself expressed as a numeric constant.

Examples: The following statement defines the value of a data area as a header to be displayed at the top of each new page written to the terminal:

```
WRITE LINE TO TERMINAL
  FROM (EMPL_HEAD) TO (END_EMPL_HEAD)
  HEADER (1);
```

The following statement writes the value in the specified data area to a new page on the terminal:

```
WRITE LINE TO TERMINAL
   NOWAIT NEWPAGE
   FROM (EMPL_RPT) LENGTH (60);
```

Status codes: Upon completion of the WRITE LINE TO TERMINAL function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4707	A logical or permanent I/O error has occurred during processing.
4731	The line request block (LRB) contains an invalid field, indicating a possible error in the program's parameters.
4732	The derived length of the specified line output area is zero or negative.
4738	The specified program variable-storage entry has not been allocated as required. A GET STORAGE request must be issued.
4743	The line I/O session has been canceled; the terminal operator has pressed CLEAR (3270s), ATTENTION (2741s), or BREAK (teletypes).

6.74 WRITE LOG (DC/UCF)

The WRITE LOG statement retrieves a predefined message from the message area of the data dictionary and optionally writes the message to a specified location in program variable storage. Retrieved messages are sent to the destination specified in the message definition; typical destinations are the operator's console and the system log file. If the operator's console has been defined as the message destination, the WRITE LOG statement can request a reply. When a reply is requested, control is not returned to the issuing task until the reply is received.

Message ID and severity code: The message ID specified in the WRITE LOG statement is a 7-digit number. The first six (most significant) digits make up the actual message ID used to retrieve the message from the data dictionary; the seventh digit is a severity code. This severity code is predefined in the dictionary and is retrieved along with the message text to indicate the action to be taken after the message is written to the log. The following table shows severity codes and corresponding system actions.

Severity code	Corresponding action by the system
0	Return control to the issuing program and continue processing.
1	Snap all task resources and return control to the issuing program.
2	Snap all system areas and return control to the issuing program.
3	Snap all task resources and abend the task with a task abend code of D002.
4	Snap all system areas and abend the task with a task abend code of D002.
5	Terminate the task with a task abend code of D002.
6	Undefined.
7	Undefined.
8	Snap all system areas and abend the system with a system abend code of 3996.
9	Terminate the system with a system abend code of 3996.

Message IDs that are not in the dictionary: If a WRITE LOG statement specifies a message ID that is not in the dictionary, the system will use a prototype message but will perform the action associated with the severity code specified in the WRITE LOG request.

Messages containing symbolic parameters: Messages stored in the data dictionary can contain symbolic parameters. Symbolic parameters, identified by an ampersand (&) followed by a 2-digit numeric identifier, can appear in any order within the message. The WRITE LOG statement can specify replacement values for one or more symbolic parameters; however, the position of replacement values within the WRITE LOG request must correspond exactly with the 2-digit numeric identifier in the message text. For example, the first value specified corresponds to &01, the second to &02, and so forth.

Syntax

```
►►─── WRITE LOG MESSAGE ID (message-id) ──────────────────────────────────►

►─┬────────────────────────────────────────────────────────────────────┬──►
  │         ┌─────────────────────────────────────────────────────┐    │
  └─ PARMS ─↓─ FROM (parm-location) ──┬─ TO (end-parm-location) ──┴────┘
                                      └─ LENGTH (parm-length) ────┘

►─┬──────────────────────────────────────────────────────────────────┬────►
  └─ REPLY INTO (reply-location) ──┬─ TO (end-reply-location) ───────┤
                                   └─ MAX LENGTH (reply-max-length) ─┘

►─┬───────────────────────────────────────────────────────────────────┬──►─
  └─ TEXT INTO (text-return-location) ──┬─ TO (end-text-return-location) ──┤
                                        └─ MAX LENGTH (text-max-length) ───┘

─►───────────────────────────────────────────────────┬─ ; ──────────────►◄
  ┌──────────────────────────────────────────────────┘
  └─┬─────────────────────────────┬─┬───────────────┬┘
    └─ MESSAGE_PREFIX ──┬─ YES ← ─┘ └─ TEXT ONLY ───┘
                        └─ NO ────┘
```

Parameters

MESSAGE ID <u>(message-id)</u>

Specifies the 7-digit message ID. The first six digits specify the ID of the message; the seventh digit specifies the message's severity code. *Message-id* is either the symbolic name of a user-defined FIXED BINARY(31) field that contains the message ID, or the ID itself expressed as a numeric constant. Message IDs 000001 through 900000 are reserved for use by the system; the WRITE LOG statement can specify any number in the range 900001 through 999999.

Note: The message length must be seven digits. The system will always interpret the last digit as the severity level. If you request message 987659 and do not code a severity level of zero (that is, 9876590) you are actually requesting that message 098765 be written to the log and that the system should be terminated with a 3996 abend code.

Note: When messages are added to the data dictionary for use with the WRITE LOG statement, they are assigned an 8-character identification number; the first two characters are DC. A request for message 987654 retrieves DC987654.

PARMS FROM <u>(parm-location)</u>

Supplies replacement values for one or more symbolic parameters stored with the message text. *Parm-location* is the symbolic name of a user-defined field that contains the program variable-storage entry of the replacement parameter.

TO (end-parm-location)
Indicates the end of the program variable-storage entry that contains the replacement parameter and is specified following the last data item in *parm-location*. *End-parm-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the replacement parameter.

LENGTH (parm-length)
Defines the length, in bytes, of the replacement parameter. *Parm-location* is either the symbolic name of a user-defined field that contains the length or the length itself expressed as a numeric constant.

The following WRITE LOG statement replaces a symbolic parameter with the contents of the FLT_NO field:

```
WRITE LOG MESSAGE ID (9000160)
    PARMS FROM (FLT_NO) TO (END_FLT_NO);
```

Each replacement parameter must begin with a 1-byte field from which the system obtains the length (in hexadecimal) of the parameter. This 1-byte field cannot be displayed. Consider the following example:

```
03  FLT_NO,
    05 FILLER       CHAR (1),
    05 FLT_PARM     CHAR (6) INIT ('AAA201'),
    05 END_FLT_NO   CHAR (1);
```

REPLY INTO (reply-location)
Specifies the program variable-storage entry of the area reserved for a reply to the message issued by the WRITE LOG request. *Reply-location* is the symbolic name of a user-defined field. The length of the reply area is determined by one of the following specifications:

TO (end-reply-location)
Indicates the end of the program variable-storage entry reserved for the reply and is specified following the last field in *reply-location*. *End-reply-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the reply.

MAX LENGTH (reply-max-length)
Defines the maximum length, in bytes, of the area reserved for the reply. *Reply-max-length* is either the symbolic name of a user-defined field that contains the length, or the length itself expressed as a numeric constant.

TEXT INTO (text-return-location)
Specifies that the contents of the named message, along with any replacement parameters, are to be written to the issuing program. *Text-return-location* is the symbolic name of a user-defined 1- to 132-character alphanumeric field that contains the program variable-storage entry to which the message text is to be returned. The length of the returned text is determined by one of the following specifications:

TO (end-text-return-location)
Indicates the end of the program variable-storage entry reserved for the text and is specified following the last data item in *text-return-location*. *End-text-return-location* is the symbolic name of either a user-defined dummy

byte field or a field that contains a data item not associated with the returned text.

MAX LENGTH (text-max-length)
Defines the maximum length, in bytes, of the program variable-storage entry reserved for the returned message text. *Text-max-length* is either the symbolic name of a user-defined field that contains the text length, or the length itself expressed as a numeric constant.

MESSAGE_PREFIX YES/NO
Specifies the format of the message prefix.

YES
Indicates that the message text is preceded by:

```
IDMS DCnnnnnnn Vssssss REPLYnn
```

DC*nnnnnnn* is the message number, V*ssssss* is the system number, and REPLY*nn* is the message's system-supplied reply number (present only if the REPLY parameter is used). YES is the default.

NO
Indicates that the message text is preceded by:

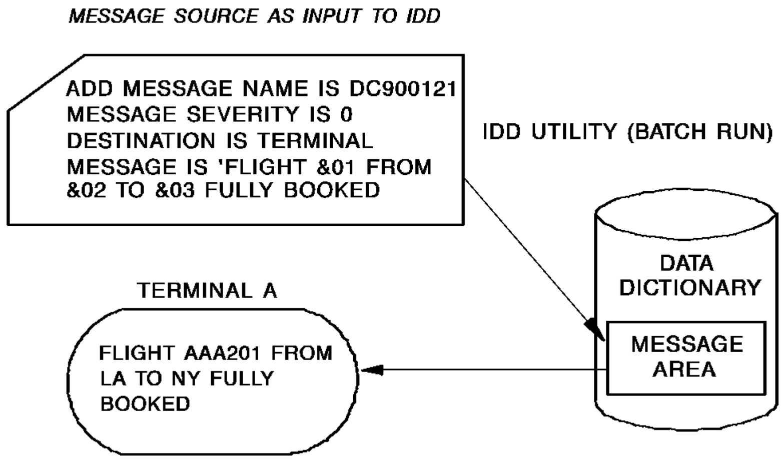
```
DCnnnnnnn
```

DC*nnnnnnn* is the message number.

TEXT ONLY
Indicates that the message text is output with no prefix.

Example: The following figure illustrates a WRITE LOG statement that supplies three replacement parameters.

Task A issues a WRITE LOG request for message 900121, specifying values to replace symbolic parameters &01, &02, and &03 stored with the message text. The system sends the message to its destination, which has been defined as the logical terminal associated with the issuing task.



WRITE LOG REQUEST

```
WRITE LOG MESSAGE ID (9001210)
PARMS FROM (FLT_NO) TO (END_FLT_NO)
      FROM (DPT_CITY) TO (END_DPT_CITY)
      FROM (ARV_CITY) TO (END_ARV_CITY);
```

```
WHERE: FLT_NO   = AAA201
       DPT_CITY = LA
       ARV_CITY = NY
```

Status codes: Upon completion of the WRITE LOG function, the ERROR_STATUS field of the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
3623	No storage or resource control element (RCE) can be allocated for the specified reply area.
3624	The maximum number of outstanding replies has been exceeded; a maximum of 98 messages can be awaiting reply at a given time.
3631	The parameter list is invalid.

6.75 WRITE PRINTER (DC/UCF)

The WRITE PRINTER statement transmits data from a task to a terminal defined to the system as a printer device during system generation. Any type of terminal can be designated as a printer; however, the terminal is usually a hard-copy device.

The system does not transmit data directly from program variable storage to the terminal. Rather, data is passed to a queue maintained by the system, and from the queue to the printer. The data stream passed to the queue by the WRITE PRINTER request contains only data; the system adds the necessary line and device control characters when it writes the data to the printer.

Note: Native mode data streams (that is, those that contain device-control information as well as user data) can also be transmitted with a WRITE PRINTER request. This capability is useful in formatting reports for 3280-type printers.

Each line of data transmitted in a WRITE PRINTER request is considered a record. Each record is associated with a *report* in the print queue. A report consists of one or more records. Any task can have up to 256 active print reports. A program can issue multiple WRITE PRINTER requests, each specifying a different report. Because the system maintains the records associated with each report individually, records associated with one report are not interspersed with records associated with other reports when printed.

WRITE PRINTER directs reports to print classes and destinations: The WRITE PRINTER request can direct reports to print classes and to destinations:

- **Print classes** — During system generation, one or more print classes are associated with each terminal designated as a printer. For each report, the first record transmitted to the print queue by means of a WRITE PRINTER request establishes the print class for that report. The report will be printed on the first available printer that is assigned the same print class.
- **Destinations** — Destinations are groups of terminals, printers, or users. If destinations have been defined during system generation, the WRITE PRINTER request can direct a report to a destination. Reports sent to printer destinations are printed on the first available printer for the destination, regardless of print class.

The system prints a report only when that report is completed, either explicitly as part of a WRITE PRINTER request or implicitly when the issuing task terminates.

Affect of termination: Normal task termination, a FINISH TASK request, or a COMMIT TASK request will end all of the task's reports. Queued reports are made eligible for printing.

Abnormal task termination (abend) or a ROLLBACK TASK request will cause any queued reports belonging to the task to be deleted.

Syntax

```
►►── WRITE PRINTER ─┬───────────┬─┬──────────┬──────────────────────────────────►
                    ├─ NEWPAGE ─┤ └─ ENDRPT ─┘
                    └─ ERASE ───┘

 ►─┬─┬────────────┬─ FROM (message-location) ─┬─ TO (end-message-location) ──┬┬─►
   │ └─ NATIVE ───┘                           └─ LENGTH (message-length) ────┘│
   └─ SCREEN CONTENTS ────────────────────────────────────────────────────────┘

 ►─┬─────────────────────────────────┬─┬──────────────────────────────────┬─►
   └─ COPIES ( ─┬─ 1 ← ──────┬─ ) ───┘ └─ REPORT ID ( ─┬─ 1 ← ─────┬─ ) ──┘
                └─ copy-count ┘                        └─ report-id ┘

 ►─┬──────────────────────────────────────────────────┬──────────────────────►
   └─┬─ CLASS (printer-class) ─────────────────────────┤
     └─ DESTINATION (printer-destination) ─┬───────┬───┘
                                           └─ ALL ─┘

 ►─┬──────────┬─┬──────────┬─ ; ─────────────────────────────────────────────►◄
   └─ HOLD ───┘ └─ KEEP ───┘
```

Parameters

NEWPAGE
Specifies that the data stream will be printed beginning on a new page. The keywords NEWPAGE and ERASE are synonymous.

ENDRPT
Indicates that the data stream constitutes the last record in the specified report. When ENDRPT is specified, the report can be printed before the issuing task has terminated. However, the program must issue a COMMIT TASK request to signal the system to print the ended report. A subsequent WRITE PRINTER request with the same report id will start a separate report.

FROM (message-location)
Specifies the program variable-storage entry of the data to be transmitted to the print queue. *Message-location* is the symbolic name of a user-defined field. The length of the data area is determined by one of the following specifications:

TO (end-message-location)
Indicates the end of the program variable-storage entry that contains the data to be transmitted to the print queue and is specified following the last data-item entry in *message-location*. *End-message-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the output data.

LENGTH (message-length)
Defines the length, in bytes, of the data stream. *Message-length* is either the symbolic name of a user-defined field that contains the length of the data, or the length itself expressed as a numeric constant.

NATIVE
Specifies that the data stream contains device-control characters. If NATIVE is not specified, the system automatically inserts the necessary characters.

SCREEN CONTENTS
Specifies (for 3270-type devices only) that the contents of the currently displayed screen are to be transmitted to the print queue. If SCREEN CONTENTS is

specified with a non-3270 terminal or a remote 3270 terminal running under TCAM, an error condition results.

COPIES (1/copy-count)
Specifies the number of copies of the report to be printed. The specified copy count must be an integer in the range 1 through 255; the default is 1. *Copy-count* is either the symbolic name of a user-defined field that contains the copy count, or the count itself expressed as a numeric constant.

REPORT ID (1/report-id)
Specifies the identifier of the report to be printed. The specified identifier must be an integer in the range 1 through 255; the default is 1. *Report-id* is either the symbolic name of a user-defined field that contains the report ID, or the ID itself expressed as a numeric constant.

CLASS (printer-class)
Specifies the print class to which the report will be assigned. Valid print classes are 1 through 64; the default is 1. *Printer-class* is either the symbolic name of a user-defined field that contains the print class, or the class itself expressed as a numeric constant.

DESTINATION (printer-destination)
Specifies the 1- to 8-character destination to which the report will be routed. *Printer-destination* is either the symbolic name of a user-defined field that contains the destination, or the destination itself enclosed in quotation marks. The specified destination must have been defined during system generation.

ALL
Specifies that the report is to be printed on all of the printers belonging to the specified destination. The report will be printed, one printer at a time, and saved until it has been printed on each of the printers associated with the destination.

CLASS/DESTINATION
Specifies a print class or destination (terminal, printer, or user). Specify this parameter only for the first line of each report. If you specify no class or destination, the default print class assigned to the issuing task's physical terminal during system generation is used.

HOLD
Specifies that a queued report will be held without being printed. The specified report will be held until a DCMT VARY REPORT *report-name* RELEASE command is issued at runtime.

KEEP
Specifies that the system will keep the report in the print queue after it has been printed. The report can be released for printing with a DCMT VARY REPORT *report-name* RELEASE command. In this way, the report can be printed several times. A KEPT report can be deleted from the print queue manually (using a DCMT VARY REPORT *report-name* DELETE command at runtime) or automatically (when the queue retention period has been exceeded).

Example: The following statement associates the data in the specified location with report 32 in the print queue and prints it beginning on a new page. Report 32 will print on the first terminal assigned to print class 3 when the program notifies the system that the report is complete or when the task terminates.

```
WRITE PRINTER
   NEWPAGE
   FROM (PASSGR_RPT) TO (END_PASSGR_RPT)
   REPORT ID (32)
   CLASS (3);
```

The following statement prints three copies of the current screen contents on a printer associated with destination A, and keeps the contents of the report in the print queue after it has printed:

```
WRITE PRINTER
   SCREEN CONTENTS
   COPIES (3)
   DESTINATION ('A')
   KEEP;
```

Status codes: Upon completion of the WRITE PRINTER function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4807	An I/O error has occurred while placing the record in the print queue.
4818	The current system definition contains no logical terminal-printer associations.
4821	The specified printer destination is undefined or is not a printer.
4831	The parameter list is invalid.
4832	The derived length of the specified printer output data area is zero or negative.
4838	The specified program variable-storage entry has not been allocated as required. A GET STORAGE request for the specified variable must be issued before the WRITE PRINTER statement.
4845	A WRITE PRINTER SCREEN CONTENTS request cannot be serviced because the terminal associated with the issuing task is not a 3270-type device or is a remote 3270 device running under TCAM.
4846	A terminal I/O error has occurred.

6.76 WRITE TERMINAL (DC/UCF)

The WRITE TERMINAL statement requests a synchronous or asynchronous data transfer from program variable storage to the terminal buffer.

Syntax

```
►►── WRITE TERMINAL ─┬──────────┬─┬─┬─ NEWPAGE ─┬──────────┬──────────►
                     ├─ WAIT ← ─┤ │ └─ ERASE ───┘          │
                     └─ NOWAIT ─┘ └─┬─ EAU ────────────────┘
                                    └─ ERASE ALL UNPROTECTED ─┘

►── FROM (output-data-location) ─┬─ TO (end-output-data-location) ─┬─ ; ──►◄
                                 └─ LENGTH (output-data-length) ───┘
```

Parameters

WAIT/NOWAIT
Indicates whether the write operation is to be synchronous or asynchronous.

WAIT
Specifies that the write operation will be synchronous; the issuing task will automatically relinquish control to the system and wait for completion of the write operation before continuing processing. WAIT is the default.

NOWAIT
Specifies that the write operation will be asynchronous; the issuing task will continue executing.

Note: If NOWAIT is specified, the program must issue a CHECK TERMINAL request (described earlier in this section) before performing any other I/O operation.

NEWPAGE/EAU
Specifies the mechanism to be used with the write operation.

NEWPAGE
Activates the page-eject (SYSINOUT devices) or erase-write (3270-type devices) mechanism to erase the contents of a screen. If NEWPAGE is not specified, the WRITE TERMINAL request will write over rather than erase data displayed on the terminal. The keywords NEWPAGE and ERASE are synonymous.

EAU
Activates (for 3270-type devices only) the erase-all-unprotected mechanism. Following a WRITE TERMINAL EAU function, only protected fields remain on the terminal. If EAU is specified, the FROM clause (described below) need not be specified.

FROM (output-data-location)
Specifies the program variable-storage entry of the output data stream. *Output-data-location* is the symbolic name of a user-defined field. The length of the output data stream is determined by one of the following specifications:

TO (end-output-data-location)
Indicates the end of the output data stream and is specified following the last data-item entry in *output-data-location*. *End-output-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the output data stream.

LENGTH (output-data-length)
Defines the length, in bytes, of the output data stream. *Output-data-length* is either the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant.

Example: The following statement illustrates an asynchronous basic mode request to write data to the terminal from the specified location in program variable storage:

```
WRITE TERMINAL
   NOWAIT
   FROM (TERM_LINE) LENGTH (72);
```

Status codes: Upon completion of the WRITE TERMINAL function, the ERROR_STATUS field in the the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4525	The output operation has been interrupted; the terminal operator has pressed ATTENTION or BREAK.
4526	A logical error (for example, an invalid control character) has been encountered in the output data stream.
4527	A permanent I/O error has occurred during processing.
4528	The dial-up line for the terminal has been disconnected.
4531	The terminal request block (TRB) contains an invalid field, indicating a possible error in the program's parameters.
4532	The derived length of the specified output data area is zero or negative.
4539	The terminal associated with the issuing task is out of service.

6.77 WRITE THEN READ TERMINAL (DC/UCF)

The WRITE THEN READ TERMINAL statement requests a transfer of data from program variable storage to the terminal buffer and, when the terminal operator has completed entering data, a transfer of that data back to program variable storage.

Syntax

```
►►── WRITE ─┬────────┬─ READ TERMIMAL ─┬──────────┬──────────────────────────►
            └─ THEN ─┘                 ├─ WAIT ← ─┤
                                       └─ NOWAIT ─┘

►─┬──────────────────────────────┬────────────────────────────────────────────►
  ├─┬─ NEWPAGE ─┬────────────────┤
  │ └─ ERASE ───┘                │
  └─┬─ EAU ──────────────────────┤
    └─ ERASE ALL UNPROTECTED ─┘

►─┬───────────────────────────────────────────────────┬───────────────────────►
  ├─ MODIFIED ─┬─ FROM POSITION (screen-position) ────┘
  └─ BUFFER ───┘

►── FROM (output-data-location) ─┬─ TO (end-output-data-location) ─────┬──────►
                                 └─ LENGTH (output-data-length) ───────┘

►── INTO (input-data-location) ─┬─ TO (end-input-data-location) ──────────┬───►
                                └─ MAX LENGTH (input-data-max-length) ────┘

►─┬──────────────────────────────────────────────────┬─ ; ──────────────────►◄
  └─ RETURN LENGTH INTO (input-data-actual-length) ──┘
```

Parameters

WAIT/NOWAIT
Indicates whether the I/O operation is to be synchronous or asynchronous.

WAIT
Specifies that the I/O operation will be synchronous; the issuing task will automatically relinquish control to the system and must wait for completion of the I/O operation before processing can continue. WAIT is the default.

NOWAIT
Specifies that the I/O operation will be asynchronous; the issuing task will continue executing.

Note: If NOWAIT is specified, the program must issue a CHECK TERMINAL request (described earlier in this chapter) before performing any other I/O operation.

NEWPAGE/EAU
Specifies the mechanism to be used with the write operation:

NEWPAGE
Activates the page-eject (SYSINOUT devices) or erase-write (3270-type devices) mechanism to erase the contents of a screen. If NEWPAGE is not specified, the WRITE TERMINAL request will write over rather than erase

data displayed on the terminal. The keywords NEWPAGE and ERASE are synonymous.

EAU
Activates (for 3270-type devices only) the erase-all-unprotected mechanism. Following a WRITE TERMINAL EAU function, only protected fields remain on the terminal. If EAU is specified, the FROM clause (described below) need not be specified.

MODIFIED/BUFFER
Transfers (for 3270-type devices only) data to the application program without requiring the terminal operator to signal completion of data entry.

MODIFIED
Reads all modified fields in the terminal buffer into program variable storage.

BUFFER
Executes a READ BUFFER command that reads the entire contents of the terminal buffer into the program variable storage.

FROM POSITION (screen-position)
Defines the buffer address (screen position) at which the read will start. *Screen-position* is either the symbolic name of a user-defined FIXED BINARY(31) field or the address itself enclosed in quotation marks.

FROM (output-data-location)
Specifies the program variable-storage entry of the output data stream. *Output-data-location* is the symbolic name of a user-defined field. The length of the output data stream is determined by one of the following specifications:

TO (end-output-data-location)
Indicates the end of the output data stream and is specified following the last data-item entry in *output-data-location*. *End-output-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the output data stream.

LENGTH (output-data-length)
Defines the length, in bytes, of the output data stream. *Output-data-length* is either the symbolic name of a user-defined field that contains the length of the data stream, or the length itself expressed as a numeric constant.

INTO (input-data-location)
Specifies the program variable-storage entry of the data area reserved for the input data stream. *Input-data-location* is the symbolic name of a user-defined field. The length of the input data stream is determined by one of the following specifications:

TO (end-input-data-location)
Indicates the end of the data area reserved for the input data stream and is specified following the last data-item entry in *input-data-location*. *End-input-data-location* is the symbolic name of either a user-defined dummy byte field or a field that contains a data item not associated with the data area reserved for the input data stream.

MAX LENGTH **(input-data-max-length)**
Defines the length, in bytes, of the data area reserved for the input data stream. *Input-data-max-length* is either the symbolic name of a user-defined field that contains the length of the data stream, or the length itself expressed as a numeric constant.

If the input data stream is larger than the data area reserved in program variable storage, the system truncates the data stream to fit the available space.

RETURN LENGTH INTO **(input-data-actual-length)**
Indicates the location to which the system will return the actual length of the input data stream. *Input-data-actual-length* is the symbolic name of a user-defined field. If the data stream has been truncated, *input-data-actual-length* contains the original length before truncation.

Example: The following statement illustrates a basic mode request to write data from the program (OUTPUT_LINE) to the terminal, read the data from the terminal to the specified location (INPUT_LINE) in the program, and return the actual length of the input data stream (LINE_LENGTH) to variable storage:

```
WRITE THEN READ TERMINAL
   WAIT
   FROM (OUTPUT_LINE) TO (END_INPUT_LINE)
   INTO (INPUT_LINE) MAX LENGTH (80)
   RETURN LENGTH INTO (LINE_LENGTH);
```

Status codes: Upon completion of the WRITE THEN READ TERMINAL function, the ERROR_STATUS field in the IDMS-DC communications block indicates the outcome of the operation:

Status code	Meaning
0000	The request has been serviced successfully.
4519	The input area specified for the return of data is too small; the returned data has been truncated to fit the available space.
4525	The output operation has been interrupted; the terminal operator has pressed ATTENTION or BREAK.
4526	A logical error (for example, an invalid control character) has been encountered in the output data stream.
4527	A permanent I/O error has occurred.
4528	The dial-up line for the terminal has been disconnected.
4531	The terminal request block (TRB) contains an invalid field, indicating a possible error in the program's parameters.
4532	The derived length of the specified I/O data area is zero or negative.

Status code	Meaning
4535	Storage for the input buffer cannot be acquired because the specified program variable-storage entry has been allocated.
4538	The specified program variable-storage entry has not been allocated and the GET STORAGE option has not been specified.
4539	The terminal device associated with the issuing task is out of service.

6.78 Logical-record clauses (WHERE and ON)

Logical-record clauses are used with any of the four DML statements that access logical records (that is, OBTAIN, MODIFY, STORE, or ERASE). The logical-record clauses are as follows:

- **WHERE** — Specifies criteria used to select and/or criteria used to limit the selection of logical-record occurrences.
- **ON** — Tests for a specific path status returned to indicate the result of a logical-record DML statement.

The following subsections describe the WHERE and ON clauses.

6.78.1 WHERE

The WHERE clause has two major functions:

- **To direct the program to a path**, predefined in the subschema by the DBA and transparent to the application program. This allows you to access the database without issuing specific instructions for navigating the database.
- **To specify selection criteria to be applied to a logical record**. This allows the program to specify attributes of the desired logical record, thereby reducing the need for the program to inspect multiple logical records to isolate the logical record of interest.

The WHERE clause is issued in the form of a boolean expression that consists of comparisons and keywords connected by boolean operators (AND, OR, and NOT). The format of the WHERE clause follows PL/I syntax rules (that is, operands or operators are separated by a blank).

Note:

If you use the WHERE clause, you must specify the 48-character set in your source program; IDMSDMLP assumes the use of the 48-character set when it generates LRF code. For further information, see Chapter 2, “DML Precompiler Options” on page 2-1.

```
►►── WHERE ( ─┬───────────┬─┬─ dba-designated-keyword ─┬──────────────►
              └─┬─ NOT ─┬─┘ └─ comparison ─────────────┘
                └─ ¬ ───┘

 ►─┬──────────────────────────────────────────────────────────┬─ ) ──────►◄
   │ ┌──────────────────────────────────────────────────────┐ │
   └─↓─┬─┬─ AND ─┬─┬─┬───────────┬─┬─ dba-designated-keyword ─┬─┴─┘
       │ └─ & ───┘ │ └─┬─ NOT ─┬─┘ └─ comparison ─────────────┘
       └─┬─ OR ──┬─┘   └─ ¬ ───┘
         └─ | ───┘
```

Expansion of comparison

```
►►─┬─ literal ──────────────────────────────────────────┬─┬─ CONTAINS ─┬─►
   │ ┌──────────┐                                        │ ├─ MATCHES ──┤
   ├─↓─ group-id.─┴─ idd-defined-variable-field-name ────┤ ├─┬─ EQ ─┬───┤
   └─ arithmetic-expression ─────────────────────────────┘ │ └─ = ──┘   │
                                                           ├─┬─ NE ─┬───┤
                                                           │ └─ ¬= ─┘   │
                                                           ├─┬─ GT ─┬───┤
                                                           │ └─ > ──┘   │
                                                           ├─┬─ LT ─┬───┤
                                                           │ └─ < ──┘   │
                                                           ├─┬─ GE ─┬───┤
                                                           │ └─ >= ─┘   │
                                                           └─┬─ LE ─┬───┘
                                                             └─ <= ─┘

 ►─┬─ literal ──────────────────────────────────────────┬──────────────►◄
   │ ┌──────────┐                                        │
   ├─↓─ group-id.─┴─ variable-field-name ────────────────┤
   │ ┌──────────┐                                        │
   ├─↓─ group-id.─┴─ logical-record-field-name ─┬────────┬┤
   │                                            └─ OF LR ─┘│
   └─ arithmetic-expression ─────────────────────────────┘
```

Parameters

<u>dba-designated-keyword</u>

Specifies a DBA-designated keyword to be applied to the logical record that is the object of the command. *Dba-designated-keyword* is a keyword specified by the DBA that is applicable to the logical record named in the command; it can be no longer than 32 characters. The keyword represents an operation to be performed at the path level and serves only to route the logical-record request to the appropriate, predetermined path.

A path must exist to service a request that issues *dba-designated-keyword.* If no such path exists, the DML precompiler flags this condition by issuing an error message.

<u>comparison</u>

Specifies a comparison operation to be performed, using the indicated operands and operators. It also serves to direct the logical-record request to a path.

Individual comparisons and keywords are connected by the boolean operators AND, OR, and NOT. Parentheses can be used to clarify a multiple-comparison boolean expression or to override the precedence of operators.

literalidd-defined-variable-field-namearithmetic-expression

Identifies a left or right comparison operand.

<u>literal</u>

Specifies a literal value. *Literal* can be any alphanumeric or numeric literal. Alphanumeric literals must be enclosed in quotation marks.

<u>idd-defined-variable-field-name</u>

Specifies a program variable storage field predefined in the dictionary. *Idd-defined-variable-field-name* must be an elementary element. It cannot be a group element. Group elements can only be used for qualification.

The optional qualifier *group-id* uniquely identifies the named variable field. This qualifier is required if *idd-defined-variable-field-name* is not unique within program variable storage. *Group-id* names the group element that contains the field. A maximum of 15 different *group-id* qualifiers can be specified to identify as many as 15 levels of group elements.

arithmetic-expression

Specifies an arithmetic expression designated as a unary minus (-), unary plus (+), simple arithmetic operation, or compound arithmetic operation. Arithmetic operators permitted in an arithmetic expression are add (+), subtract (-), multiply (*), and divide (/). Operands can be literals, variable-storage fields, and logical-record fields as described above. On the left side of the comparison you cannot use a key value.

CONTAINS/MATCHES/EQ/NE/GT/LT/GE/LE

Specifies the comparison operator. Operators are evaluated in the following order:

1. Comparisons enclosed in parentheses
2. Arithmetic, comparison, and boolean operators by order of precedence, from highest to lowest:
   a. Unary plus or minus in an arithmetic expression
   b. Multiplication or division in an arithmetic expression
   c. Addition or subtraction in an arithmetic expression
   d. MATCHES or CONTAINS comparison operators
   e. EQ, NE, GT, LT, GE, LE comparison operators
   f. NOT boolean operator
   g. AND boolean operator
   h. OR boolean operator
3. From left to right within operators of equal precedence

CONTAINS

Is true if the value of the right operand occurs in the value of the left operand. Both operands included with the CONTAINS operator must be alphanumeric values and elementary elements.

MATCHES

Is true if each character in the left operand matches a corresponding character in the right operand (the mask). When MATCHES is specified, LRF compares the left operand with the mask, one character at a time, moving from left to right. The result of the match is either true or false: the result is true if the end of the mask is reached before encountering a character in the left operand that does not match a corresponding character in the mask. The result is false if LRF encounters a character in the left operand that does not match a mask character.

Three special characters can be used in the mask to perform pattern matching: **@**, which matches any alphabetic character; **#**, which matches any numeric character; and *****, which matches any alphabetic or numeric character. Both the left operand and the mask must be alphanumeric values and elementary elements.

EQ
Is true if the value of the left operand is equal to the value of the right operand.

NE
Is true if the value of the left operand is not equal to the value of the right operand.

GT
Is true if the value of the left operand is greater than the value of the right operand.

LT
Is true if the value of the left operand is less than the value of the right operand.

GE
Is true if the value of the left operand is greater than or equal to the value of the right operand.

LE
Is true if the value of the left operand is less than or equal to the value of the right operand.

<u>logical-record-field-name</u>
Specifies a data field that participates in the named logical record. *Logical-record-field-name* must be an elementary element. It cannot be a group element. Group elements can only be used for qualification.

The optional qualifier *group-id* uniquely identifies the named logical-record field. This qualifier is required if *logical-record-field-name* is not unique within all subschema records, including those that are not part of the logical record, and all non CA-IDMS/DB records copied into the program. *Group-id* names the group element or database record that contains the field. A maximum of 15 different *group-id* qualifiers can be specified to identify as many as 15 levels of group elements.

The optional OF LR parameter specifies that the value of the named field at the time that the request is issued will be used throughout processing of the request. If the value of the field changes during request processing, LRF will continue to use the original value. If the OF LR entry is not included and the value of the field changes during request processing, the new field value in variable storage will be used if the field is required for further request processing.

Usage of the WHERE clause: If the WHERE clause compares a CALC-key field to a *literal*, the literal's format must correspond exactly to the CALC-key definition. Enclose the literal in quotation marks if the CALC key has a usage of DISPLAY, and use leading zeros if the literal consists of fewer characters than the field's picture. For example, if the *calc-key-field* CALC key is defined as CHAR (3), code the WHERE clause as follows:

```
WHERE (calc-key-field) EQ '054';
```

The WHERE clause can contain as many comparisons and keywords as required to specify the criteria to be applied to the logical record. If necessary, the value of the SIZE parameter in the INCLUDE IDMS SUBSCHEMA_LR_CTRL statement can be increased to accommodate very large and complex WHERE clause specifications. Processing efficiency is not affected by the composition of the WHERE clause (other than the logical order of the operators, as noted below), since LRF automatically uses the most efficient path to process the logical-record request.

Examples: The following logical-record request uses a DBA-designated keyword (PROGRAMMER_ANALYSTS) to direct LRF to a DBA-defined access path:

```
OBTAIN NEXT RECORD (EMP_JOB_LR)
    WHERE (PROGRAMMER_ANALYSTS);
```

The following logical-record request uses boolean selection criteria to specify the desired occurrence of EMP_JOB_LR:

```
OBTAIN RECORD (EMP_JOB_LR)
    WHERE (OFFICE_CODE_0450 EQ '001');
```

6.78.2 ON clause

The ON clause tests for a specific path status returned to indicate the result of the statement. If LRF returns the specified path status, the imperative statement included in the ON clause is executed; if the specified path status is not returned, the imperative statement included in the ON clause is ignored and IDMS_STATUS is performed.

The ON clause tests for a standard or DBA-defined path status, which is in the form of a 1- to 16-character unquoted string. Path statuses are issued during execution of the path selected to service the request.

Standard path statuses: Standard path statuses are as follows:

- **LR_FOUND** — Returned when the logical-record request has been successfully executed. This status can be returned as the result of any of the four LRF DML statements. When LR_FOUND is returned, the ERROR_STATUS field in the IDMS communications block contains 0000.

- **LR_NOT_FOUND** — Returned when the logical record specified cannot be found, either because no such record exists or because all such occurrences have already been retrieved. This status can be returned as the result of any of the four LRF DML statements, provided that the path to which LRF is directed includes retrieval logic. When LR_NOT_FOUND is returned, the ERROR_STATUS field in the IDMS communications block contains 0000.

  Note: A successful STORE can return LR_NOT_FOUND if its WHERE clause references a logical-record field and the STORE path performs no OBTAINs.

- **LR_ERROR** — Returned when a logical-record request is issued incorrectly or when an error occurs in the processing of the path selected to service the request.

When LR_ERROR is returned, the type of error-status code returned to the program in the ERROR_STATUS field in the IDMS-DB communications block differs according to the type of error:

- When the error occurs in the **logical-record request**, the ERROR_STATUS field contains an error-status code issued by LRF (major code of 20).
- When an error occurs in **logical-record path processing**, the ERROR_STATUS field contains an error-status code issued by the DBMS (major code from 00 to 19). For more information on error-status codes, refer to Chapter 3, "Communications Blocks and Error Detection" on page 3-1.

Syntax

```
►►── ON LR_STATUS (path-status)  imperative-statement;──────────────────────►◄
```

Parameters

path-status
Names the path status that will be tested. *Path-status* must be a 1- to 16-character alphanumeric value.

imperative-statement
Specifies the program action to be taken if the indicated path status results from the logical-record request.

Example: The following statements use the path status LR_NOT_FOUND in two different ways. If LR_NOT_FOUND occurs following the initial statement, an LR_MISSING message is output; if LR_NOT_FOUND occurs in subsequent statements, an END_OF_LR message is output.

```
OBTAIN FIRST RECORD (EMP_JOB_LR)
   WHERE (OFFICE_CODE_0450 EQ OFFICE_CODE_IN);
   ON LR_STATUS (LR_NOT_FOUND)
      CALL LR_MISSING;
 .
 .
 .
OBTAIN NEXT RECORD (EMP_JOB_LR)
   WHERE (OFFICE_CODE_0450 EQ OFFICE_CODE_IN);
   ON LR_STATUS (LR_NOT_FOUND)
      CALL END_OF_LR;
 .
 .
 .
CALL OBTAIN_REST_LR;
```

Status codes: The following codes are returned to the ERROR_STATUS field in the IDMS-DB or IDMS-DC communications block when an LR_ERROR path status is returned to the LR_STATUS field in the LRC block:

Status code	Meaning
2001	The requested logical record was not found in the subschema. (The path DML statement, EVALUATE, returns 0000 if true, and 2001 if false.)
2008	The named record is not in the subschema, or the specified request is not permitted for the named record.
2010	The subschema prohibits access to logical records.
2018	A path command has attempted to access a database record that has not been bound.
2040	The WHERE clause in an OBTAIN NEXT command directed LRF to a different processing path than did the WHERE clause in the preceding OBTAIN command for the same logical record.
2041	The request's WHERE clause cannot be matched to a path in the subschema.
2042	The logical-record path for the request specifies return of the LR_ERROR status.
2043	Bad or inconsistent data was encountered in the logical-record buffer during evaluation of the request's WHERE clause.
2044	The request's WHERE clause does not include data required by the logical-record path.
2045	A subscript value in a WHERE clause is either less than zero or greater than its maximum allowed value.
2046	A program check has revealed an arithmetic exception (for example, overflow, underflow, significance, divide) during evaluation of a WHERE clause.
2063	The request's WHERE clause contains a keyword that exceeds the 16-character maximum.
2064	The path command has attempted to access a CALC data item that has not been defined properly in the subschema.
2072	The request's WHERE clause is too long to be evaluated in the available work area.

Appendix A. DML Precompile, PL/I Compile, and Link-Edit JCL

A.1 About this appendix

This appendix presents the JCL used to prepare PL/I source code that contains DML statements. Link-edit considerations are also discussed. JCL samples are included.

A.2 Compiling a PL/I program

To compile a PL/I program under the DML precompiler:

1. Execute the program IDMSDMLP
2. Execute the PL/I compiler
3. Link edit

Input to IDMSDMLP consists of source code written in PL/I and DML, protocol/control information, and dictionary record descriptions. Output from IDMSDMLP includes:

- A source PL/I program
- A DML source listing and diagnostics

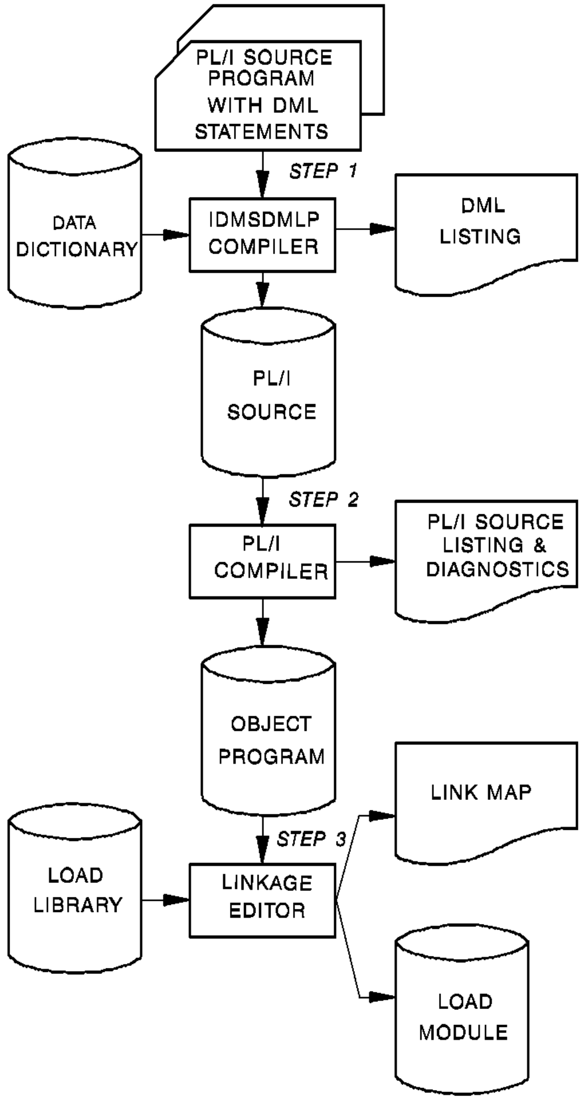
Input to the PL/I compiler consists of the source program produced by IDMSDMLP. Output includes:

- An object program
- PL/I listings

Input to the linkage editor consists of the object program produced by the PL/I compiler. Output includes:

- A load module
- A link-edit map

The following figure illustrates the steps involved in compiling a PL/I program.



The JCL used to compile and link edit the DMLP source statements under the CA-IDMS/DB central version are shown in this appendix. Local mode considerations are noted where appropriate.

Note: IBM PL/I compilers running under VSE/ESA do not generate reentrant code. Accordingly, if your applications are large, multiple user deadlocks may result because of space limitations.

A.2.1 Under OS/390

Executing under the central version: **IDMSDMLP (central version) (OS/390)**

```
//*****************************************************************
//**                  PRECOMPILE PL/I PROGRAM                    **
//*****************************************************************
//precomp  EXEC PGM=IDMSDMLP,REGION=1024K,
//             PARM='optional parameters'
//STEPLIB  DD   DSN=idms.dba.loadlib,DISP=SHR
//         DD   DSN=idms.loadlib,DISP=SHR
//sysctl   DD   DSN=idms.sysctl,DISP=SHR
//dcmsg    DD   DSN=idms.sysmsg.ddldcmsg,DISP=SHR
//SYS001   DD   UNIT=disk,SPACE=(TRK,(10,10))
//SYS002   DD   UNIT=disk,SPACE=(TRK,(10,10))
//SYS003   DD   UNIT=disk,SPACE=(TRK,(10,10))
//SYSPCH   DD   DSN=&&source,DISP=(NEW,PASS),
//             UNIT=disk,SPACE=(TRK,(10,5),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSLST   DD   SYSOUT=A
//SYSIDMS  DD   *
DMCL=dmcl-name
DICTNAME=dictionary-name
Other SYSIDMS parameters, as appropriate
/*
//SYSIPT   DD   *
PL/I DML source statements
/*
//*****************************************************************
//**                  COMPILE PL/I PROGRAM                       **
//*****************************************************************
//plicmp   EXEC PGM=IEL0AA,REGION=300K,
//             PARM='DECK,LIST,OFFSET,STORAGE,NOP'
//STEPLIB  DD   DSN=sys1.pliopt,DISP=SHR
//SYSUT1   DD   UNIT=disk,SPACE=(1024,(200,50),,CONTIG,ROUND),
//             DCB=BLKSIZE=6144
//SYSPUNCH DD   DSN=&&object,DISP=(NEW,PASS),
//             UNIT=disk,SPACE=(TRK,(10,5),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSN=&&source,DISP=(OLD,DELETE)
//*****************************************************************
//**                  LINK PROGRAM MODULE                        **
//*****************************************************************
//link     EXEC PGM=IEWL,REGION=300K,PARM='LET,LIST,XREF'
//SYSUT1   DD   UNIT=disk,SPACE=(TRK,(20,5))
//SYSLIB   DD   DSN=sys1.plibase,DISP=SHR
//loadlib  DD   DSN=idms.loadlib,DISP=SHR
//SYSLMOD  DD   DSN=user.loadlib,DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSN=&&object,DISP=(OLD,DELETE)
//         DD   *
 INCLUDE loadlib(IDMS)      Required, except omit for CICS
 INCLUDE loadlib(IDMSCANC)  Required for BATCH and DC_BATCH
                            if using IDMS_STATUS module
 INCLUDE loadlib(IDMSOPTI)  Optional; BATCH and DC_BATCH only
 INCLUDE loadlib(IDMSCINT)  Required for CICS, otherwise omit
 ENTRY   userentry
 NAME    userprog(R)
/*
//*
```

precomp	Name of the precompile step
optional parameters	LIST/NOLIST determines whether or not a DML source listing is generated. DMLIST/NODMLIST in the source code overrides this parameter.

	DICTNAME specifies the dictionary you want to access. DICTNAME can also be specified as a SYSIDMS parameter.
	DEBUG=CARD causes each input record from source to be written to SYSLST as it is processed. This allows you to identify any records that may cause a processing loop.
	SCHEMA = *schema-name* specifies the default schema-name qualifier for the precompiler to use when processing an INCLUDE TABLE statement that does not supply a qualifier.
	NOINSTALL specifies that the precompiler should only check syntax.
	SQL=NO/89/FIPS specifies the SQL syntax standard that the precompiler should apply when checking the validity of SQL statements in the program. Option NO is the default; means that compliance with a named SQL standard is not checked or enforced, and all CA-IDMB/DB extentions are permitted. Option 89 directs the precompiler to use ANSI X3.135-1989 (Rev), Database Language SQL with integrity enhancement as the standard for compliance. Option FIPS directs the precompiler to use FIPS PUB 127-1, *Database Language SQL* as the standard for compliance.
	DATE=ISO/USA/EUR/JIS specifies the format of the DATE data type to be used for communication between the program and the database when the access module is executed.
	TIME=ISO USA EUR/JIS specifies the format of the TIME data type to be used for communication between the program and the database when the access module is executed. For information on EXEC PGM parameters that are applicable to SQL access, refer to the *CA-IDMS SQL Programming* document.
`idms.dba.loadlib`	Dataset name of the load library containing the DMCL and database name table load modules
`idms.loadlib`	Dataset name of the load library containing CA-IDMS executable modules
`sysctl`	DDname of SYSCTL file

idms.sysctl	Dataset name of SYSCTL file
dcmsg	DDname of the dictionary message area
idms.sysmsg.ddldcmsg	Dataset name of the dictionary message area (DDLDCMSG)
&&source	Name of temporary dataset output from DML precompiler
link	Name of the link edit step
dmcl-name	Name of the DMCL to access at runtime
dictionary-name	Name of the dictionary to access at runtime
plicmp	Name of the PL/I compile step
sys1.pliopt	PL/I system library
&&object	Name of temporary dataset output from PL/I compiler
disk	Symbolic device name
sys1.plibase	Name of the load library containing subroutines that may need to be linked to the user program
user.loadlib	Name of a user application load library
userentry	Program entry point
userprog	Name of the user program in the load library

Note: Depending on the CV operating environment, an IDMSOPTI module link edited with IDMSDMLP can be used in place of or in addition to the SYSCTL file.

EXEC PGM parameters: You can specify LIST/NOLIST, DICTNAME, or DEBUG=CARD as IDMSDMLP parameters, or you can optionally include your site-specific parameters in an Assembler module (having the CSECT name of EDBPPARM) and link that module with IDMSDMLP. For example:

```
EDBPPARM
   DC 'NOLIST'
   DC X'00'
   END
```

Runtime parameters: To specify a dictionary or DMCL to access at runtime, you can include DICTNAME and DMCL parameters in a SYSIDMS DD statement in the JCL (see previous sample JCL).

►► For complete information on SYSIDMS runtime parameters, refer to *CA-IDMS Database Administration*.

Executing in local mode:

IDMSDMLP (local mode) (OS/390)

```
//*****************************************************************
//**                  PRECOMPILE PL/I PROGRAM                     **
//*****************************************************************
//precomp  EXEC PGM=IDMSDMLP,REGION=1024K,
//             PARM='optional parameters'
//STEPLIB  DD   DSN=idms.dba.loadlib,DISP=SHR
//         DD   DSN=idms.loadlib,DISP=SHR
//dictb    DD   DSN=idms.appldict.ddldml,DISP=SHR
//dcmsg    DD   DSN=idms.sysmsg.ddldcmsg,DISP=SHR
//sysjrnl  DD   DSN=idms.tapejrnl,DISP=(NEW,CATLG),UNIT=tape
//SYS001   DD   UNIT=disk,SPACE=(TRK,(10,10))
//SYS002   DD   UNIT=disk,SPACE=(TRK,(10,10))
//SYS003   DD   UNIT=disk,SPACE=(TRK,(10,10))
//SYSPCH   DD   DSN=&&source,DISP=(NEW,PASS),
//             UNIT=disk,SPACE=(TRK,(10,5),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSLST   DD   SYSOUT=A
//SYSIDMS  DD   *
DMCL=dmcl-name
DICTNAME=dictionary-name
Other SYSIDMS parameters, as appropriate
/*
//SYSIPT   DD   *
PL/I DML source statements
/*
//*****************************************************************
//**                  COMPILE PL/I PROGRAM                        **
//*****************************************************************
//plicmp   EXEC PGM=IEL0AA,REGION=300K,
//             PARM='DECK,LIST,OFFSET,STORAGE,NOP'
//STEPLIB  DD   DSN=sys1.pliopt,DISP=SHR
//SYSUT1   DD   UNIT=disk,SPACE=(1024,(200,50),,CONTIG,ROUND),
//             DCB=BLKSIZE=6144
//SYSPUNCH DD   DSN=&&object,DISP=(NEW,PASS),
//             UNIT=disk,SPACE=(TRK,(10,5),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSN=&&source,DISP=(OLD,DELETE)
//*****************************************************************
//**                  LINK PROGRAM MODULE                         **
//*****************************************************************
//link     EXEC PGM=IEWL,REGION=300K,PARM='LET,LIST,XREF'
//SYSUT1   DD   UNIT=disk,SPACE=(TRK,(20,5))
//SYSLIB   DD   DSN=sys1.plibase,DISP=SHR
//loadlib  DD   DSN=idms.loadlib,DISP=SHR
//SYSLMOD  DD   DSN=user.loadlib,DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSN=&&object,DISP=(OLD,DELETE)
//         DD   *
 INCLUDE loadlib(IDMS)      Required, except omit for CICS
 INCLUDE loadlib(IDMSCANC)  Required for BATCH and DC_BATCH
                            if using IDMS_STATUS module
 INCLUDE loadlib(IDMSOPTI)  optional; BATCH and DC_BATCH only
 INCLUDE loadlib(IDMSCINT)  Required for CICS, otherwise omit
 ENTRY   userentry
 NAME    userprog(R)
/*
//*
```

`dictdb`	DDname of the application dictionary definition area
`idms.appldict.ddldml`	Dataset name of the application dictionary definition area

sysjrnl	DDname of the tape journal file
idms.tapejrnl	Dataset name of the tape journal file
tape	Symbolic device name

►► For information about other variables, see the table below the JCL for central version.

A.2.2 Under VSE/ESA

Executing under the central version: IDMSDMLP (VSE/ESA)

```
* step1
// EXEC PROC=IDMSLBLS
// UPSI b
// DLBL     idmspch,'temp.dmlp',0
// EXTENT   sys020,nnnnnn,,,ssss,llll
// ASSGN    sys020,DISK,VOL=nnnnnn,SHR
// EXEC      IDMSDMLP
DMCL=dmcl-name
DICTNAME=dictionary-name
Other optional SYSIDMS parameters
/*
PL/I DML source statements
/*
* step2
// DLBL      IJSYSIN,'temp.dmlp',0
// EXTENT    SYSIPT,nnnnnn
   ASSGN     SYSIPT,DISK,VOL=nnnnnn,SHR
// OPTION    CATAL,NODECK,NOSYM
  PHASE userprog,*
// EXEC      PL/I
* step3
   CLOSE     SYSIPT,SYSRDR
ENTRY (dmlp)
// EXEC      LNKEDT
/*
```

Note: You can define a SYSCTL file in the JCL to override the IDMSOPTI statement for the back-end system:

```
// DLBL    sysctl,'idms.sysctl',,DA
// EXTENT  sys008,nnnnnn
// ASSGN   sys008,DISK,VOL=nnnnnn,SHR
```

IDMSLBLS	Procedure containing all of the file definitions required by the system ►► For a complete listing of IDMSLBLS, see "IDMSLBLS procedure", later in this section.
b	Appropriate UPSI switch, 1-8 characters, if specified in the IDMSOPTI module
idmspch	Filename of dataset output from the DML precompiler
temp.dmlp	File ID of the dataset output from the DML precompiler

sys020	Logical unit assignment of DMLP output
nnnnnn	Volume serial identifier of appropriate disk volume
dmcl-name	Name of the DMCL to access at runtime
dictionary-name	Name of the dictionary to access at runtime
ssss	Starting track (CKD) or block (FBA) of disk extent
llll	Number of tracks (CKD) or blocks (FBA) of disk extent
userprog	Name of program in the library
dmlp	Name of PL/I DML module
sysctl	Filename of the SYSCTL file
idms.sysctl	File ID of the SYSCTL file
sys008	Logical unit assignment of the SYSCTL file

SYSIDMS parameters: You can use SYSIDMS parameters to specify information about your runtime environment. The SYSIDMS parameters DICTNAME and DMCL are used in this JCL stream.

►► For information on other optional SYSIDMS parameters, refer to *CA-IDMS Database Administration.*

Output to disk or tape file: To route punched output to a sequential disk file or to a tape file, use a SYSPCH statement in the JCL.

Executing in local mode: To execute IDMSDMLP in local mode:

- Remove the UPSI statement
- Add the following statements in the IDMSDMLP step:

```
// TLBL    sysjrnl,'idms.tapejrnl',,nnnnnn,,f
// ASSGN   sys009,TAPE,VOL=nnnnnn
```

sysjrnl	Filename of the tape journal file
idms.tapejrnl	File ID of the tape journal file
f	File number of the tape journal file
sys009	Logical unit assignment for journal file

INCLUDE statements: Provide INCLUDE statements in local mode or central version JCL as follows. Place the following statements in the second step, before EXEC PL/I:

```
ACTION NOAUTO              Prevents multiple inclusions of IDMS

INCLUDE IDMS               IDMS interface for use with COMRG

INCLUDE IDMSOPTI           You can omit IDMSOPTI for local mode

INCLUDE IDMSCANC           Local mode abort entry point
                           (omit IDMSCANC if TP application)

INCLUDE IDMSCINT           For CICS only, replaces INCLUDE IDMS
```

IDMSLBLS procedure: IDMSLBLS is a procedure that contains file definitions for the dictionaries, sample databases, disk journal files, and SYSIDMS file provided during installation.

You can tailor the following IDMSLBLS procedure (provided at installation) to reflect the filenames and definitions in use at your site. Reference IDMSLBLS as shown in the previous VSE/ESA JCL job stream.

```
  ──────── LIBDEFS ────────
// LIBDEF  *,SEARCH=idmslib.sublib
// LIBDEF  *,CATALOG=user.sublib
/*  ───────────────────────── LABELS ─────────────────────────
// DLBL    idmslib,'idms.library',1999/365
// EXTENT  ,nnnnnn,,,ssss,1500
// DLBL    dccat,'idms.system.dccat',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,31
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dccatl,'idms.system.dccatlod',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,6
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dccatx,'idms.system.dccatx',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,11
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dcdml,'idms.system.ddldml',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,101
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dclod,'idms.system.ddldclod',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,21
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dclog,'idms.system.ddldclog',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,401
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dcrun,'idms.system.ddldcrun',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,68
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dcscr,'idms.system.ddldcscr',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,135
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dcmsg,'idms.sysmsg.ddldcmsg',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,201
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dclscr,'idms.sysloc.ddlocscr',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,6
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dirldb,'idms.sysdirl.ddldml',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,201
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dirllod,'idms.sysdirl.ddldclod',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,2
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    empdemo,'idms.empdemo1',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,11
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    insdemo,'idms.insdemo1',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,6
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    orgdemo,'idms.orgdemo1',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,6
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    empldem,'idms.sqldemo.empldemo',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,11
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    infodem,'idms.sqldemo.infodemo',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,6
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    projdem,'idms.projseg.projdemo',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,6
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    indxdem,'idms.sqldemo.indxdemo',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,6
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    sysctl,'idms.sysctl',1999/365,SD
// EXTENT  SYSnnn,nnnnnn,,,ssss,2
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    secdd,'idms.sysuser.ddlsec',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,26
```

```
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dictdb,'idms.appldict.ddldml',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,51
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    dloddb,'idms.appldict.ddldclod',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,51
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    sqldd,'idms.syssql.ddlcat',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,101
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    sqllod,'idms.syssql.ddlcatl',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,51
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    sqlxdd,'idms.syssql.ddlcatx',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,26
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    asfdml,'idms.asfdict.ddldml',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,201
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    asflod,'idms.asfdict.asflod',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,401
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    asfdata,'idms.asfdict.asfdata',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,201
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    ASFDEFN,'idms.asfdict.asfdefn',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,101
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    j1jrnl,'idms.j1jrnl',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,54
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    j2jrnl,'idms.j2jrnl',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,54
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    j3jrnl,'idms.j3jrnl',1999/365,DA
// EXTENT  SYSnnn,nnnnnn,,,ssss,54
// ASSGN   SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL    SYSIDMS,'#SYSIPT',0,SD
/+
/*
```

idmslib.sublib	Name of the sublibrary within the library containing CA-IDMS modules
user.sublib	Name of the sublibrary within the library containing user modules
idmslib	Name of the file containing CA-IDMS modules
idms.library	ID associated with the file containing CA-IDMS modules
SYSnnn	Logical unit of the volume for which the extent is effective
nnnnnn	Volume serial identifier of appropriate disk volume
ssss	Starting track (CKD) or block (FBA) of disk extent
dccat	Filename of the system dictionary catalog (DDLCAT) area
idms.system.dccat	ID of the system dictionary catalog (DDLCAT) area

dccatl	Filename of the system dictionary catalog load (DDLCATLOD) area
idms.system.dccatlod	ID of the system dictionary catalog load (DDLCATLOD) area
dccatx	Name of the system dictionary catalog index (DDLCATX) area
idms.system.dccatx	ID of the system dictionary catalog index (DDLCATX) area
dcdml	Name of the system dictionary definition (DDLDML) area
idms.system.ddldml	ID of the system dictionary definition (DDLDML) area
dclod	Name of the system dictionary definition load (DDLDCLOD) area
idms.system.ddldclod	ID of the system dictionary definition load (DDLDCLOD) area
dclog	Name of the system log area (DDLDCLOG) area
idms.system.ddldclog	ID of the system log (DDLDCLOG) area
dcrun	Name of the system queue (DDLDCRUN) area
idms.system.ddldcrun	ID of the system queue (DDLDCRUN) area
dcscr	Name of the system scratch (DDLDCSCR) area
idms.system.ddldcscr	ID of the system scratch (DDLDCSCR) area
dcmsg	Name of the system message (DDLDCMSG) area
idms.sysmsg.ddldcmsg	ID of the system message (DDLDCMSG) area
dclscr	Name of the local mode system scratch (DDLOCSCR) area
idms.sysloc.ddlocscr	ID of the local mode system scratch (DDLOCSCR) area
dirldb	Name of the IDMSDIRL definition (DDLDML) area
idms.sysdirl.ddldml	ID of the IDMSDIRL definition (DDLDML) area
dirllod	Name of the IDMSDIRL definition load (DDLDCLOD) area
idms.sysdirl.dirllod	ID of the IDMSDIRL definition load (DDLDCLOD) area
empdemo	Name of the EMPDEMO area
idms.empdemo1	ID of the EMPDEMO area
insdemo	Name of the INSDEMO area

idms.insdemo1	ID of the INSDEMO area
orgdemo	Name of the ORGDEMO area
idms.orgdemo1	ID of the ORDDEMO area
empldem	Name of the EMPLDEMO area
idms.sqldemo.empldemo	ID of the EMPLDEMO area
infodem	Name of the INFODEMO area
idms.sqldemo.infodemo	ID of the INFODEMO area
projdem	Name of the PROJDEMO area
idms.projseg.projdemo	ID of the PROJDEMO area
indxdem	Name of the INDXDEMO area
idms.sqldemo.indxdemo	ID of the INDXDEMO area
sysctl	Name of the SYSCTL file
idms.sysctl	ID of the SYSCTL file
secdd	Name of the system user catalog (DDLSEC) area
idms.sysuser.ddlsec	ID of the system user catalog (DDLSEC) area
dictdb	Name of the application dictionary definition area
idms.appldict.ddldml	ID of the application dictionary definition (DDLDML) area
dloddb	Name of the application dictionary definition load area
idms.appldict.ddldclod	ID of the application dictionary definition load (DDLDCLOD) area
sqldd	Name of the SQL catalog (DDLCAT) area
idms.syssql.ddlcat	ID of the SQL catalog (DDLCAT) area
sqllod	Name of the SQL catalog load (DDLCATL) area
idms.syssql.ddlcatl	ID of SQL catalog load (DDLCATL) area
sqlxdd	Name of the SQL catalog index (DDLCATX) area
idms.syssql.ddlcatx	ID of the SQL catalog index (DDLCATX) area
asfdml	Name of the asf dictionary definition (DDLDML) area
idms.asfdict.ddldml	ID of the asf dictionary definition (DDLDML) area
asflod	Name of the asf dictionary definition load (ASFLOD) area
idms.asfdict.asflod	ID of the asf dictionary definition load (ASFLOD) area
asfdata	Name of the asf data (ASFDATA) area

idms.asfdict.asfdata	ID of the asf data area (ASFDATA) area
ASFDEFN	Name of the asf data definition (ASFDEFN) area
idms.asfdict.asfdefn	ID of the asf data definition area (ASFDEFN) area
j1jrnl	Name of the first disk journal file
idms.j1jrnl	ID of the first disk journal file
j2jrnl	Name of the second disk journal file
idms.j2jrnl	ID of the second disk journal file
j3jrnl	Name of the third disk journal file
idms.j3jrnl	ID of the third disk journal file
SYSIDMS	Name of the SYSIDMS parameter file

A.2.3 Under VM/ESA

Executing under the central version: IDMSDMLP (VM/ESA)

```
FILEDEF SYSIPT DISK sysipt data a (RECFM F LRECL ppp BLKSIZE nnn
FILEDEF SYSPCH DISK prgnme PL/I A
FILEDEF SYSIDMS DISK sysidms parms a (RECFM F LRECL ppp BLKSZE nnn
EXEC IDMSFD
OSRUN IDMSDMLP PARM='CVMACH=vmid'     DML precompile step
FILEDEF TEXT DISK prgnme TEXT A
GLOBAL TXTLIB plilibvs IDMSLIB1
PL/I prgnme (OSDECK APOST LIB         PL/I compile step
TXTLIB DEL utextlib prgnme
TXTLIB ADD utextlib prgnme
FILEDEF SYSLMOD uloadlib LOADLIB a (RECFM V LRECL 1024 BLKSIZE 1 024
FILEDEF objlib1 DISK IDMSLIB1 TXTLIB A
FILEDEF objlib DISK utextlib TXTLIB a
FILEDEF SYSLIB DISK plilibvs TXTLIB p
LKED linkctl (LIST XREF LET MAP RENT NOTERM PRINT SIZE 512K 64K
                                      Link edit step
```

sysipt data a	Filename, filetype, and filemode of the file that contains PL/I DML source statements
ppp	Record length of the data file
nnn	Blocksize of the data file
prgnme	Filename of the PL/I program
sysidms parms a	Filename, filetype, and filemode of the file that contains SYSIDMS parameters (parameters that define your runtime environment)
vmid	ID of the virtual machine running the CA-IDMS/DB central version
plilibvs	Filename of the library that contains PL/I logic modules
utextlib	Filename of the user text library

uloadlib	Filename of the user load library
objlib1	DDname of the first CA-IDMS/DB object library
objlib	DDname of the user object library
plilibvs	Filename of the library that contains PL/I logic modules
linkctl	Filename of the file that contains the linkage editor control statements

How to edit the SYSIDMS file: To edit the SYSIDMS file, enter these VM/ESA commands:

```
XEDIT sysidms parms a (NOPROF
INPUT
 .
 .
 .
SYSIDMS parameters
 .
 .
 .
FILE
```

To run IDMSDMLP, include the DMCL and DICTNAME SYSIDMS parameters.

►► For information on SYSIDMS, refer to *CA-IDMS Database Administration*.

How to create the SYSIPT file: To create the SYSIPT file, enter these VM/ESA commands:

```
XEDIT sysipt data a (NOPROF
INPUT
 .
 .
 .
DML source statements
 .
 .
 .
FILE
```

How to create the LINKCTL file: To create the LINKCTL file, enter these VM/ESA commands:

```
XEDIT linkctl data a (NOPROF
INPUT
INCLUDE objlib(prgnme)
INCLUDE objlib1(IDMS)      IDMS is required, omit for CICS
INCLUDE objlib1(IDMSCINT) IDMS is required for CICS only
INCLUDE objlib1(IDMSCANC) IDMSCANC for BATCH and DC_BATCH
ENTRY prgnme
NAME prgnme(R)
FILE
```

Executing in local mode: To execute IDMSDMLP in local mode, remove the CVMACH parameter from OSRUN, and do *one* of the following:

- Link IDMSDMLP with an IDMSOPTI program that specifies local execution mode
- Specify *LOCAL* as the first input parameter in the file specified in the FILEDEF SYSIPT statement
- Modify the OSRUN statement, as follows:

  ```
  OSRUN IDMSDMLP PARM='*LOCAL*'
  ```

  Note: This option is valid only if the OSRUN command is issued from a System Product Interpreter or from an EXEC2 file.

A.3 Link-edit considerations

The modules involved in the link edit of an application program contain six external references. Some must be resolved depending on the mode of operation. Check unresolved references against the following table to ensure proper linkage to the program.

Reference	Referenced by	Resolved by	Comments
ABORT	Application	IDMSCANC	Should be resolved
IDMS	Application	IDMS	Must be resolved
IDMSOPTI[1]	IDMS	IDMSOPTI module	Must be resolved under OS/390 if using the central version without a SYSCTL file, and under VSE/ESA if using the central version
IDMSWAIT[1]	IDMS	IDMSWAIT	Must be resolved if user-written wait program is desired; otherwise, system routine is used

[1]Under OS/390, IDMSOPTI is a weak external reference (WXTRN).

Appendix B. Call Formats

B.1 About this appendix

This appendix contains the call formats used by CA-IDMS/DB and CA-IDMS/DC to execute DML commands. Each DML function can be coded using standard CALL statements.

The tables in this appendix present the function codes and arguments that are passed to CA-IDMS/DB and CA-IDMS/DC for execution of a DML command.

About arguments 0 and 1: Note the following information about arguments 0 and 1 when you review the tables in this appendix:

- **Argument 0** — This argument is passed for all functions. It contains SUBSCHEMA-CTRL, the IDMS-DB or IDMS-DC communications block.
- **Argument 1** — CA-IDMS/DB passes the IDBMSCOM array as argument 1. CA-IDMS/DC passes the DCBMSCOM array as argument 1.

Example of a call format: The following example shows the expanded call format for a BIND RECORD statement (BIND EMPLOYEE):

```
CALL 'IDMS' (SUBSCHEMA_CTRL
                  ,IDBMSCOM (48)
                  ,'EMPLOYEE      '
                  ,EMPLOYEE;
                                   );
```

Order of expansions: CA-IDMS/DB call expansions are presented first, CA-IDMS/DC expansions second. Formats are grouped in different tables according to statement function.

B.2 CA-IDMS/DB call formats

CA-IDMS/DB passes the IDBMSCOM array as argument 1.

Arguments marked with asterisks have default values.

B.2.1 Control statements

Major function code	Database statement (in PL/I DML)	(1) (nn)	Calling arguments (2)	(3)	(4)	(5)
14	BIND RUN_UNIT	59	IDMS-DB communications block*	subschema-name*		
	BIND RUN_UNIT FOR subschema-name	59	IDMS-DB communications block*	subschema-name		
	BIND RUN_UNIT NODENAME nodename	59	IDMS-DB communications block*	subschema-name*	subschema-control* or subschema-lr-control*	node name
	BIND RUN_UNIT FOR subschema-name NODENAME nodename	59	IDMS-DB communications block*	subschema-name	subschema-control* or subschema-lr-control*	node name
	BIND RUN_UNIT FOR subschema-name DBNAME database-name	59	IDMS-DB communications block*	subschema-name	subschema-control* or subschema-lr-control*	node name
	BIND RUN_UNIT NODENAME nodename DBNAME database-name	59	IDMS-DB communications block*	subschema-name*	subschema-control* or subschema-lr-control*	node name
	BIND RUN_UNIT FOR subschema-name NODENAME nodename DBNAME database-name	59	IDMS-DB communications block*	subschema-name	subschema-control* or subschema-lr-control*	node name
	BIND record-name	48	record-id	record-location*		
	BIND record-name TO record-location	48	record-id	record-location		
	BIND record-location WITH record-name	48	record-id	record-location		
	BIND PROCEDURE FOR procedure-name TO procedure-control-location	73	procedure-name	procedure-control-location		

09	READY	37				
	READY area-name	37	area-name			
	READY area-name USAGE-MODE IS RETRIEVAL	37	area-name			
	READY area-nameUSAGE-MODE IS PROTECTED RETRIEVAL	39	area-name			
	READY area-name USAGE-MODE IS EXCLUSIVE RETRIEVAL	40	area-name			
	READY area-name USAGE-MODE IS UPDATE	36	area-name			
	READY area-name USAGE-MODE IS PROTECTED UPDATE	38	area-name			
	READY area-name USAGE-MODE IS EXCLUSIVE UPDATE	41	area-name			
	READY USAGE-MODE IS... ** Choose function code from 36-41, as shown above	**				
01	FINISH	02				
18	COMMIT	66				
	COMMIT ALL	95				
19	ROLLBACK	67				
	ROLLBACK CONTINUE 96					
06	KEEP CURRENT	87				
	KEEP EXCLUSIVE CURRENT	88				
	KEEP CURRENT record-name	89	record-name			
	KEEP EXCLUSIVE CURRENT record-name	90	record-name			
	KEEP CURRENT WITHIN set-name	91	set-name			
	KEEP EXCLUSIVE CURRENT WITHIN set-name	93	set-name			
	KEEP CURRENT WITHIN area-name	93	area-name			
	KEEP EXCLUSIVE CURRENT WITHIN area-name	94	area-name			

16	IF set-name IS EMPTY...	64	set-name			
	IF set-name IS NOT EMPTY... (Upon return to user run-unit, the Error Status indicator = '0000' if set is empty; '1601' if not empty.)	65	set-name			
	IF set-name MEMBER...	60	set-name			
	IF NOT set-name MEMBER... (Upon return to user run-unit, the Error Status indicator = '0000' if the record (current of run-unit) is linked into the specified set; '1601' if it is not a member.)	62	set-name			

B.2.2 Modification statements

Major function code	Database statement (in PL/I DML)	(1) (nn)	Calling arguments (2)	(3)	(4)	(5)
12	STORE record-name	42	record-name			
07	CONNECT record-name TO set-name	44	record-name	set-name		
08	MODIFY record-name	35	record-name			
11	DISCONNECT record-name FROM set-name	46	record-name	set-name		
02	ERASE record-name	52	record-name			
	ERASE record-name PERMANENT MEMBERS	03	record-name			
	ERASE record-name SELECTIVE MEMBERS	53	record-name			
	ERASE record-name ALL MEMBERS	04	record-name			

B.2.3 Retrieval statements

Major function code	Database statement (in PL/I DML)	Calling arguments (1) (nn)	(2)	(3)	(4)	(5)
03	FIND DB-KEY db-key	75	db-key			
	FIND record-name DB-KEY IS db-key	06	record-name	db-key		
	FIND DB-KEY db-key PAGE_INFO page-info	29	dbkey	page-info		
	FIND CURRENT	30				
	FIND CURRENT record-name	07	record-name			
	FIND CURRENT WITHIN set-name	08	set-name			
	FIND CURRENT WITHIN area-name	09	area-name			
	FIND NEXT WITHIN set-name	14	set-name			
	FIND NEXT record-name WITHIN set-name	10	record-name	set-name		
	FIND PRIOR WITHIN set-name	16	set-name			
	FIND PRIOR record-name WITHIN set-name	12	record-name	set-name		
	FIND FIRST WITHIN set-name	20	set-name			
	FIND FIRST record-name WITHIN set-name	18	record-name	set-name		
	FIND LAST WITHIN set-name	24	set-name			
	FIND LAST record-name WITHIN set-name	22	record-name	set-name		
	FIND sequence-number WITHIN set-name	78	set-name	sequence-number		
	FIND sequence-number record-name WITHIN set-name	76	record-name	set-name	sequence-number	
	FIND sequence-number WITHIN set-name	78	set-name	sequence-number		
	FIND sequence-number record-name WITHIN set-name	76	record-name	set-name	sequence-number	
	FIND NEXT WITHIN area-name	15	area-name			
	FIND NEXT record-name WITHIN area-name	11	record-name	area-name		
	FIND PRIOR WITHIN area-name	17	area-name			
	FIND PRIOR record-name WITHIN area-name	13	record-name	area-name		

FIND FIRST WITHIN area-name	21	area-name			
FIND FIRST record-name WITHIN area-name	19	record-name	area-name		
FIND LAST WITHIN area-name	25	area-name			
FIND LAST record-name WITHIN area-name	23	record-name	area-name		
FIND sequence-number WITHIN area-name	79	area-name	sequence-number		
FIND sequence-number record-name WITHIN area-name	77	record-name	area-name	sequence-number	
FIND sequence-number WITHIN area-name	79	area-name	sequence-number		
FIND sequence-number record-name WITHIN area-name	79	record-name	area-name c.sequence-number		
FIND OWNER WITHIN set-name	31	set-name			
FIND CALC(ANY) record-name	32	record-name			
FIND DUPLICATE record-name	50	record-name			
FIND record-name WITHIN set-name USING sort-field -name	33	record-name	set-name	sort-field-name	
FIND record-name WITHIN set-name CURRENT USING sort-field-name	51	record-name	set-name	sort-field-name	
OBTAIN (any of the above FIND record selection expressions). Call generated consists of arguments described above for the FIND in question plus an additional argument of IDBMSCOM (43) function. For example: OBTAIN CALC record	32	record-name	IDBMSCOM (43)		
OBTAIN PRIOR record-name WITHIN set-name KEEP/KEEP EXCLUSIVE (any of the above FIND/OBTAIN record selection expressions). Call generated consists of arguements described above for the FIND/OBTAIN in question plus one of the following additional IDMSCOM functions: KEEP.....IDBMSCOM(87) KEEP EXCLUSIVE.....IDBMSCOM(88) For example: OBTAIN KEEP CALC record-name	12 32	record-name record-name	 IDBMSCOM (43)	 IDBMSCOM (87)	
FIND KEEP EXCLUSIVE CURRENT	30	IDBMSCOM (88)			

05	GET	43				
	GET record-name	34	record-name			
17	RETURN db-key FROM index-set-name CURRENCY KEY INTO symbolic-key	81	index-set-name	db-key	symbolic-key	
	RETURN db-key FROM index-set-name FIRST KEY INTO symbolic-key	82	index-set-name	db-key	symbolic-key	
	RETURN db-key FROM index-set-name LAST KEY INTO symbolic-key	83	index-set-name	db-key	symbolic-key	
	RETURN db-key FROM index-set-name NEXT KEY INTO symbolic-key	84	index-set-name	db-key	symbolic-key	
	RETURN db-key FROM index-set-name PRIOR KEY INTO symbolic-key	85	index-set-name	db-key	symbolic-key	
	RETURN db-key FROM index-set-name USING index-key-value KEY INTO symbolic-key	86	index-set-name	db-key	index-key-value	symbolic-key

B.2.4 ACCEPT statements

Major function code	Database statement (in PL/I DML)	(1) (nn)	Calling arguments (2)	(3)	(4)	(5)
15	ACCEPT db-key FROM CURRENCY	54	db-key			
	ACCEPT db-key FROM record-name CURRENCY	55	record-name	db-key		
	ACCEPT db-key FROM set-name CURRENCY	57	set-name	db-key		
	ACCEPT db-key FROM area-name CURRENCY	56	area-name	db-key		
	ACCEPT db-key FROM set-name NEXT CURRENCY	68	set-name	db-key		
	ACCEPT db-key FROM set-name PRIOR CURRENCY	69	set-name	db-key		
	ACCEPT db-key FROM set-name OWNER CURRENCY	70	set-name	db-key		
	ACCEPT db-statistics FROM IDMS-STATISTICS	71	db-statistics			
	ACCEPT bind-address FROM record-name BIND	72	record-name	bind-address		
	ACCEPT procedure-control-location FROM procedure-name PROCEDURE	74	procedure-name	procedure-control-location		
	ACCEPT page-info-location FOR record-name	28	record-name	page-info-location		

B.2.5 LRF DML statements

Major function code	Database statement (in PL/I DML)	Calling arguments (1) (nn)	(2)	(3)	(4)	(5)
20	OBTAIN FIRST logical-record-name	99	subschema-lr-ctrl*	logical-record-location*		
	OBTAIN FIRST logical-record-name INTO alt-logical-record-location	99	subschema-lr-ctrl*	alt-logical-record-location*		
	OBTAIN NEXT logical-record-name	99	subschema-lr-ctrl*	logical-record-location*		
	OBTAIN NEXT logical-record-name INTO alt-logical-record-location	99	subschema-lr-ctrl*	alt-logical-record-location*		
	MODIFY logical-record-name	99	subschema-lr-ctrl*	logical-record-location*		
	MODIFY logical-record-name FROM alt-logical-record-location	99	subschema-lr-ctrl*	alt-logical-record-location*		
	STORE logical-record-name	99	subschema-lr-ctrl*	logical-record-location*		
	STORE logical-record-name FROM alt-logical-record-location	99	subschema-lr-ctrl*	alt-logical-record-location*		
	ERASE logical-record-name	99	subschema-lr-ctrl*	logical-record-location*		
	ERASE logical-record-name FROM alt-logical-record-location	99	subschema-lr-ctrl*	alt-logical-record-location*		

*To differentiate between the LRF DML statements, the DML precompiler places the name of the verb issued into the LRC block (*subschema-lr-ctrl*).

B.3 CA-IDMS/DC call formats

CA-IDMS/DC passes the DCBMSCOM array as argument 1.

Note: CA-IDMS/DC also passes information in the DCSTR, DCFLG, and DCNUM fields of the SUBSCHEMA-CTRL block.

B.3.1 Program management statements

Major function code	Communications statement (in PL/I DML)	(1) (nn)	Calling arguments (2)	(3)	(4)	(5)
30	TRANSFER CONTROL	23	DCFLG1	DCSTR2	parameter	
30	DC RETURN	19				
34	LOAD TABLE	15	01-level-program-location	end-01-level-program-location		
34	DELETE TABLE	5	01-level-program-location			
33	SET ABEND EXIT (STAE)	20				
33	ABEND	1				

B.3.2 Storage management statements

Major function code	Communications statement (in PL/I DML)	(1) (nn)	Calling arguments (2)	(3)	(4)	(5)
32	GET STORAGE	13	01-level-storage-data-location	end-storage-data-location		
32	FREE STORAGE	10	01-level-storage-data-location	start-free-storage-location		

B.3.3 Task management statements

Major function code	Communications statement (in PL/I DML)	(1) (nn)	Calling arguments (2)	(3)	(4)	(5)
37	ATTACH	3				
37	CHANGE PRIORITY	4				
39	ENQUEUE	9	DCFLG1	DCBMSCOM (mode)	DCBMSCOM (length)	resource -id ...
39	DEQUEUE	8	DCFLG1	DCBMSCOM (length)	resource-id	
31	WAIT	24	ecb			
31	POST	16	ecb			

B.3.4 Time management statements

Major function code	Communications statement (in PL/I DML)	(1) (nn)	Calling arguments (2)	(3)	(4)	(5)
35	GET TIME	14	return-time	return-date		
35	SET TIMER	21	start-task-data-location	end-start-task-data-location		
35	SET TIMER (post)	21	post-ecb			

B.3.5 Scratch management statistics

Major function code	Communications statement (in PL/I DML)	(1) (nn)	Calling arguments (2)	(3)	(4)	(5)
43	PUT SCRATCH	18	scratch-data-location	end-scratch-data-location		
43	GET SCRATCH	12	return-scratch-data-location	end-scratch-data-location		
43	DELETE SCRATCH	7				

B.3.6 Queue management statements

Major function code	Communications statement (in PL/I DML)	Calling arguments (1) (nn)	(2)	(3)	(4)	(5)
44	PUT QUEUE	17	queue-data-location	end-queue-data-location		
44	GET QUEUE	11	return-queue-data-location	end-queue-data-location		
44	DELETE QUEUE	6				

B.3.7 Terminal management statements

Major function code	Communications statement (in PL/I DML)	(1) (nn)	Calling arguments (2)	(3)	(4)	(5)
45	READ TERMINAL	30	input-data-location	end-input-data-location		
45	WRITE TERMINAL	30	output-data-location	end-output-data-location		
45	WRITE THEN READ TERMINAL	30	output-data-location	end-output-data-location	input-data-location	end-input-data-location
45	CHECK TERMINAL	31	input-data-location	end-input-data-location		
47	READ LINE FROM TERMINAL	32	input-data-location	end-input-data-location		
47	WRITE LINE TO TERMINAL	32	output-data-location	end-output-data-location		
47	END LINE TERMINAL SESSION	32				
48	WRITE PRINTER	37	message-location	end-message-location		
46	MAP IN (IO)	34	MRB-mapname			
46	MAP IN (NOIO)	34	MRB-mapname	mapped-data-location	end-mapped-data-location	
46	MAP IN (paging) (a)	34	MRB-mapname	datafield-name	sequence-field-name	page-number
46	MAP IN (paging) (b)	34	MRB-mapname	key	page-number	
46	MAP OUT (IO)	34	MRB-mapname	message-text	end-message-data-location or DCBMSCOM (length)	
46	MAP IN (NOIO)	34	MRB-mapname	mapped-data-location	end-mapped-data-location	
46	MAP OUT (paging)	34	MRB-mapname	message-text	end-message-data-location or DCBMSCOM (length)	key
46	MAP OUTIN	34	MRB-mapname	message-text	end-message-data-location or DCBMSCOM (length)	

46	MODIFY MAP	93	MRB-mapname	MRE	MRB-FLDLST	
46	INQUIRE MAP (a)	92	MRB-mapname	MRE		
46	INQUIRE MAP (b)	92	MRB-mapname			
46	INQUIRE MAP (c)	92	MRB-mapname	MRE		
46	INQUIRE MAP (d)	92	MRB-mapname	MRB-FLDLST		
46	STARTPAGE	40	MRB-mapname			
46	ENDPAGE	41				

B.3.8 Utility statements

		Calling arguments				
Major function code	Communications statement (in PL/I DML)	(1) (nn)	(2)	(3)	(4)	(5)
48	ACCEPT	2	return-location			
40	SNAP	22	DCSTR1	DCSTR1 (6) begin-dump-location	DCSTR1 (7) end-dump-location	title (8) DCBMSCOM (1)
49	SEND MESSAGE	38	user-id	message-location	end-message-location	
38	BIND TRANSACTION STATISTICS	28				
38	ACCEPT TRANSACTION STATISTICS	28	return-statistics-data-location			
38	END TRANSACTION STATISTICS	28	return-statistics-data-location			
51	KEEP LONGTERM	29	record-name set-name area-name			
36	WRITE LOG	25	text-return-location	end-text-return-location	reply-location (6) parameter-location	end-reply-location (7) end-parameter-location

B.3.9 Recovery statements

Major function code	Communications statement (in PL/I DML)	Calling arguments (1) (nn)	(2)	(3)	(4)	(5)
50	COMMIT	66				
50	COMMIT TASK	27				
50	FINISH	02				
50	FINISH TASK	27				
50	ROLLBACK	67				
50	ROLLBACK TASK	27				
50	WRITE JOURNAL	26	record-location	end-record-location		

B.3.10 DC_BATCH statement

Major function code	Communications statement (in PL/I DML)	Calling arguments (1) (nn)	(2)	(3)	(4)	(5)
14	BIND_TASK	28	DCSTR2			

Appendix C. Keywords

C.1 About this appendix

This appendix contains a list of keywords recognized by the DML precompiler, including words applicable in the CA-IDMS/DC environment only. All keywords marked with an asterisk are also ***reserved*** words. Reserved words cannot be used for user-defined element, record, set, procedure, or area names.

Note: The method of parsing used by the IDMSDMLP preprocessor is significantly different in CA-IDMS release 12.0 and later releases from that used in prior releases. The current parsing method looks at individual words in the source code. If it encounters a keyword, it assumes that the keyword should be expanded and tries to do so. Invalid use of reserved words can thus result in either coding errors or syntax errors. For example, if you use FIND as a variable, the parser will try to handle it as the DML verb FIND.

```
*ABEND
 ABORT
*ACCEPT
 AID
 ALARM
 ALL
 ALPHAMERIC
 ALWAYS
 ANY
 AREA
 ASSIGN
 AT
*ATTACH
 ATTRIBUTES
 BACKPAGE
 BACKSCAN
*BIND
 BLINK
 BLUE
 BRIGHT
 BROWSE
 BUFFER
 BUT
 BY
 CALC
*CALL
 CANCEL
*CHANGE
 CHANGED
*CHECK
 CLASS
 CLEAR
 CODE
*COMMIT
 COMP
 COMP_3
*CONNECT
 CONTENTS
 CONTINUE
 CONTROL
 INTERNAL
 INTERVAL
 INTO
 INVOKED
 IO
 IS
 JOURNAL
 JUSTIFY
*KEEP
 KEY
 LAST
 LEAVE
 LEFT
 LENGTH
 LEVELS
 LINE
 LINK
*LINKAGE
 LIST
 LITERALS
*LOAD
 LOCK
 LOG
 LONG
 LONGTERM
 LR
 LSSC_NODN
 LTERM
 MANUAL
*MAP
 MAP_BINDS
 MAP_CONTROL
 MAP_CONTROLS
 MAP_RECORDS
 MAPS
 MAX
 MDT
 MEMBER
 MEMBERS
 MESSAGE
*REMARKS
 REPLACE
 REPLY
 REPORT
 REQUIRED
 REREAD
 RESETKBD
 RESETMDT
 RESUME
 RETENTION
 RETURNKEY
 RETRIEVAL
 RETRY
*RETURN
 REVERSE_VIDEO
 REVERSED
 REWIND
 RIGHT
*ROLLBACK
 RUN
 RUN_UNIT
*SCHEMA
 SCRATCH
 SCREEN
 SCREENSIZE
 SECONDS
*SECTION
*SELECT
 SELECTIVE
*SEND
 SEQUENCE
 SEQUENCE-NUMBER
 SESSION
*SET
 SHARE
 SHARED
 SHORT
 SKIP
 SKIP1
 SKIP2
```

```
 COPIES
*COPY
 CORRECT
 CURRENCY
 CURRENT
 CURSOR
 DARK
*DATA

 DATABASE_KEY
 DATASTREAM
 DATE
 DB
 DB_KEY
 DBNAME
*DC
 DEBUG
*DECLARATIVES
*DELETE
*DEQUEUE
 DEST
 DESTINATION
 DETAIL
 DETECT
 DFLD
*DISCONNECT
 DISP
 DISPLAY
 DIVISION

 DUMP
 DUPLICATE
 EAU
 ECHO
 EDIT
 EJECT
 EMPTY
*END
 ENDPAGE
 ENDRPT
*ENQUEUE
*ENTRY
*ENVIRONMENT
*ERASE
 ERROR
 EVENT
 EXCEPT
 EXCLUSIVE
 EXIT
 EXITS
 EXTENDED
 EXTERNAL
 EXTRANEOUS
 FIELD
 FIELDS
 FILE
*FIND
*FINISH
 FIRST
 MODE
 MODIFIED
*MODIFY
 MODULE
 MOVE
 MRB_FLDLST
 NAME
 NATIVE

 NEWPAGE
 NEXT
 NLCR
 NO
 NOALARM
 NOBACKPAGE
 NOBACKSCAN
 NOBLINK
 NOCOLOR
 NODEADLOCK
 NODENAME
 NODUMP
 NOIO
 NOKBD
 NOLOCK
 NOMDT
 NONE
 NOPAD
 NOPRT
 NORETURN

 NORMAL
 NORMAL_VIDEO
 NOSPAN
 NOT
*NOTE
 NOTIFICATION
 NOTIFY
 NOUNDERSCORE
 NOWAIT
 NOWRITE
 NULL
 NUMERIC
*OBTAIN
 OF
 OFF
 ON
 ONLY
*OPEN
 OPTIONAL
 OUT
 OUTIN
 OUTPUT
 OWNER
 PAD
 PAGE
 PAGE_INFO
 PAGING
 PARMS
 PERMANENT
 SKIP3
 SNAP
 SOME
 SPAN
 STANDARD
 START
 STARTPAGE
 STARTPRT
 SQL
 STATISTICS
 STGID
*STOP
 STORAGE
*STORE
 SUBSCHEMA_AREANAMES
 SUBSCHEMA_BINDS
 SUBSCHEMA_CONTROL
 SUBSCHEMA_CTRL
 SUBSCHEMA_DESCRIPTION
 SUBSCHEMA_DML-LR-
  DESCRIPTION
 SUBSCHEMA_LR-CONTROL
 SUBSCHEMA_LR-CTRL
 SUBSCHEMA_LR-
  DESCRIPTION
 SUBSCHEMA_LR-NAMES
 SUBSCHEMA_LR-RECORDS
 SUBSCHEMA_NAMES
 SUBSCHEMA_RECNAMES
 SUBSCHEMA_RECORD_BINDS
 SUBSCHEMA_RECORDS
 SUBSCHEMA-SETNAMES
 SUBSCHEMA_SSNAME
 SYSTEM
 SYSVERSION
 TABLE
 TASK
 TEMPORARY
 TERMINAL
 TEST
 TEXT
 THEN
 TIME
 TIMEOUT
 TIMER
 TITLE
 TO
 TRACE
 TRANSACTION
*TRANSFER
 TRUNCATED
 TURQUOISE
 TYPE
 UNDERSCORE
 UNFORMATTED
 UNPROTECTED
 UPDATE
 UPGRADE
 USAGE_MODE
```

```
 FOR
*FREE
 FROM
*GET
 GREEN
 HEADER
 HOLD
 I_O
*ID
*IDENTIFICATION
 IDMS
*IDMS_CONTROL
 IDMS_RECORDS
 IDMS_STATISTICS
*IF
 IGNORED
 IN
 INCREMENTED
 INPUT
*INQUIRE
 INTENT
 PINK
 POSITION
*POST
 PREFIX
 PRINTER
 PRIOR
 PRIORITY
 PRIVACY
*PROCEDURE
 PROGRAM
*PROGRAM_ID
 PROTECTED
 PROTOCOL
 PTERM
*PUT
 QUEUE
*READ
*READY
 RECORD
 RED
 REDISPATCH
 RELEASE
 USER
 USING
 VALUE
 VERSION
*WAIT
 WCC
 WHERE
 WHITE
 WITH
 WITHIN
*WORKING_STORAGE
*WRITE
 XCTL
 YELLOW
 YES
 40CR
 64CR
 80CR
```

Appendix D. Notes to Teleprocessing Monitor Users

D.1 About this appendix

This appendix describes special considerations relating to application programs running under teleprocessing (TP) monitors supported by DC/UCF systems (that is, CICS, INTERCOMM, SHADOW, and TASK/MASTER).

While there are no special coding requirements for TP-monitor transactions, the following guidelines should be adhered to:

- DML statements should be coded so that all database requests (for example, BIND, READY, OBTAIN, FINISH) are executed together whenever possible to achieve maximum efficiency and ease of recovery.
- For each TP monitor, you should check with the DBA to determine the operating mode (protocol) installed. The proper mode must then be specified in the MODE clause of the DECLARE SUBSCHEMA statement.
- The DML precompiler should be executed before the TP-monitor precompiler.
- For CICS, INTERCOMM, and SHADOW applications, the mode, as installed, may require the inclusion of additional statements in each program. These requirements and the applicable modes are outlined in the following table.

  Note: The same rules apply to the INCLUDE IDMS statements used to insert logical-record source code components into the program: SUBSCHEMA_CTRL, SUBSCHEMA_LR_CTRL, and SUBSCHEMA_LR_RECORDS should be copied into the program (except under CICS_EXEC, components should be copied into the program).

<table>
<tr><th>TP monitor</th><th>If mode is...</th><th>Code these statements</th></tr>
<tr><td>CICS</td><td>CICS_STANDARD</td><td><pre> *DECLARE
 1 TWA BASED (TPTR),
   3 FILLER,
   3 INCLUDE IDMS(SUBSCHEMA_CTRL),
   3 INCLUDE
     IDMS(SUBSCHEMA_RECORDS),
   ADDRESS TWA(TPTR);
                or
 **INCLUDE IDMS(SUBSCHEMA_CTRL);
   INCLUDE IDMS(SUBSCHEMA_RECORDS);

(A CICS GETMAIN must be issued for
the SUBSCHEMA_CTRL and for
each RECORD being copied.)

   INCLUDE IDMS(IDMS_WAIT);</pre></td></tr>
<tr><td>CICS</td><td>CICS_EXEC</td><td><pre>INCLUDE IDMS(SUBSCHEMA_CTRL);
INCLUDE IDMS(SUBSCHEMA_RECORDS);</pre></td></tr>
<tr><td>INTERCOMM</td><td>INTERCOMM</td><td><pre>INCLUDE IDMS(SUBSCHEMA_CTRL);
INCLUDE IDMS(SUBSCHEMA_RECORDS);</pre></td></tr>
</table>

TP monitor	If mode is...	Code these statements
SHADOW	SHADOW	`INCLUDE IDMS(SUBSCHEMA_CTRL);` `INCLUDE IDMS(SUBSCHEMA_RECORDS);`

* If SUBSCHEMA_CTRL, SUBSCHEMA_RECORDS, and additional data does not exceed 4,096 bytes

** If SUBSCHEMA_CTRL, SUBSCHEMA_RECORDS, and additional data exceeds 4,096 bytes

Appendix E. Sample Programs and Database Definition

E.1 About this appendix

This appendix contains:

- CA-IDMS/DC programming considerations
- A sample PL/I batch program
- A sample PL/I online program
- A sample database definition - The EMPLOYEE database

The sample programs access the EMPLOYEE database. The database is shown in a diagram at the end of this appendix.

E.2 CA-IDMS/DC programming considerations

These programming considerations consist of PL/I-specific details relevant to designing CA-IDMS/DC programs:

- Reentrant code is program code that does not modify itself during program execution. CA-IDMS/DC multithreads all task requests through a single copy of a reentrant program. The CA-IDMS/DC default for PL/I programs is reentrant. To ensure that your program is reentrant, it must be compiled with the REENTRANT option of the PROCEDURE statement. Some PL/I compilers do not support reentrancy. If your compiler does not support reentrancy, your programs must be declared to CA-IDMS/DC as NONREENTRANT.
- Use the COUNT and REPORT execution options to capture statistics in the CA-IDMS/DC log. You can use these statistics to optimize storage requirements and to analyze program performance.
- Avoid using GET STORAGE repeatedly for relatively small areas when most tasks in the system are accessing larger areas. It may be more advantageous to declare PL/I variables explicitly and allow CA-IDMS/DC and PL/I to manage the storage. Internal management of storage for PL/I declared variables is handled in the same way under IDMS/DC as it is in the batch environment, with one exception. When PL/I code would normally issue an operating system request for storage, CA-IDMS/DC satisfies the request from the storage pool. Once a block of storage is allocated, it is managed as described in the PL/I programmer's guide for your installation.
- Use the REPORT execution option to determine the amount of storage actually used during program execution. Use the report statistics to set the ISA SIZE for the program in the CA-IDMS/DC system generation.
- The PL/I COUNT and FLOW options can be used to gather the following statistics:
  - The number of times each procedure is called
  - The amount of storage used during PL/I program execution.

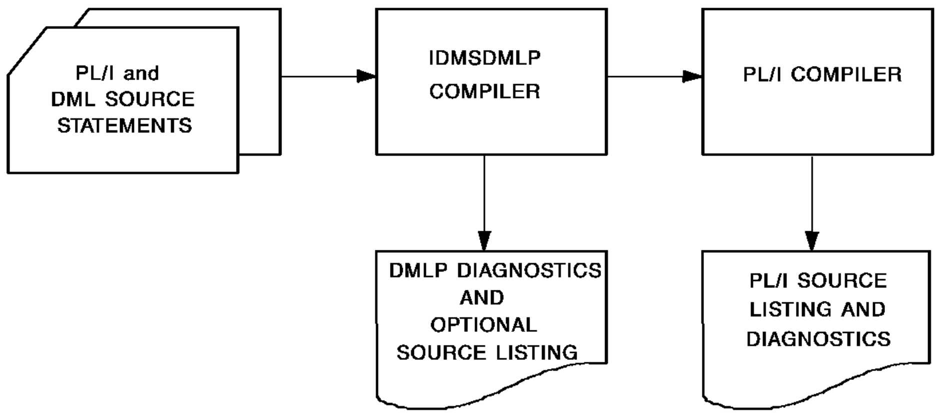
  To use these options, refer to the PL/I programmer's guide for your installation. The following considerations apply to the use of these options under CA-IDMS/DC:

  - The statistics are written to the CA-IDMS/DC system log rather than to an external file. The statistics record type is MESSAGES.
  - The statistics are not written to the log if the program terminates execution with an IDMS_DC RETURN statement. The program must use the PL/I RETURN statement. After statistics are written to the log, CA-IDMS/DC passes control to the next higher program in the transaction thread, as if an CA-IDMS/DC RETURN had been coded.
  - The REPORT and COUNT options should not be used together, since the COUNT option adds storage overhead. Accordingly, report statistics would not be accurate.

– The REPORT and COUNT options are not intended to be used in a production environment. Their use adds considerable storage and CPU overhead under CA-IDMS/DC, just as it would in a batch environment. Once the statistics have been gathered, these options should be removed from the program.

E.3 Sample batch program

The following PL/I batch program accesses database records using navigational DML statements. The following figure shows the program as it appears in the various stages of the compilation process. You create a program using PL/I and DML statements. This program is input to the DML precompiler, which produces a listing that contains diagnostics and, optionally, DML source statements. The expanded code is input to the PL/I compiler, which generates a listing of the fully expanded code and diagnostics.



E.3.1 Batch input to the DML precompiler

The following is sample batch input to the DML precompiler for PL/I.

```
//SYSIPT   DD   *
 /*RETRIEVAL*/
 /*DMLIST*/
 /*NO_ACTIVITY_LOG*/
 /*SCHEMA_COMMENTS*/
 DEPTRPT: PROC OPTIONS (MAIN) REORDER;
                                /* DECLARE SUBSCHEMA AND MODE */
 DCL    (EMPSS01 SUBSCHEMA, EMPSCHM SCHEMA VERSION 100)
                                    MODE (BATCH) DEBUG;

                              /* REQUIRED DECLARATIVES */
 DCL    IDMS ENTRY OPTIONS(INTER,ASM);
 DCL    ABORT ENTRY OPTIONS(INTER,ASM);
 DCL    ADDR BUILTIN;

       /* CONSTANTS */
 DCL    DEPT_HEADER         CHAR (11)  INIT ('DEPT REPORT');
 DCL    1 HEAD_LINE,
         5 HEAD_DEPT_ID     CHAR (9) INIT ('DEPT ID  '),
         5 HEAD_EMP_ID      CHAR (8) INIT ('EMP ID  '),
         5 HEAD_LNAME       CHAR (17) INIT ('LAST NAME        '),
         5 HEAD_FNAME       CHAR (10) INIT ('FIRST NAME');

 DCL    PRTHEAD             CHAR (44) DEFINED HEAD_LINE;

       /* LOGICAL CONSTANTS */
 DCL    YES                 BIT(1)    INIT ('1'B);
 DCL    NO                  BIT(1)    INIT ('0'B);
 DCL    EOF                 BIT(1)    INIT ('0'B);

 DCL   1 PROGRAM_FLAGS,
        5 DB_END_OF_SET     BIT(1)    INIT ('0'B);

        /* FILE DECLARATIONS */
 DCL    INFILE  FILE RECORD INPUT ENV (F BLKSIZE(80));
 DCL    OUTFILE FILE RECORD OUTPUT ENV (F RECSIZE(133) CTLASA );
 DCL    SYSPRINT FILE PRINT;
        /*   THE FOLLOWING RECORDS ARE DEFINED THROUGH IDD.       */
        /*  THE DML PRECOMPILER AUTOMATICALLY CONVERTS HYPHENS  */
        /*   TO UNDERSCORES.                                     */

 INCLUDE IDMS (DEPT-IN-REC);
 INCLUDE IDMS (PRT-OUT-REC);
        /* REDEFINE PRT_OUT_REC */
 DCL    PRTREC                  CHAR (44) DEFINED PRT_OUT_REC;

 DCL     1 PRINT_AREA,
           5 CC                 CHAR (1),
           5 PRINT_LINE         CHAR (132);

 DCL     1 SPACES               CHAR (132) INIT ( (132) ' ');

        /* POSSIBLE VALUES FOR CC                           */
 DCL     1 CONTROL_CHARACTERS,
```

```
          5 NEW_PAGE           CHAR (1) INIT ('1'),
          5 SINGLE_SPACE       CHAR (1) INIT (' '),
          5 DOUBLE_SPACE       CHAR (1) INIT ('0'),
          5 TRIPLE_SPACE       CHAR (1) INIT ('-'),
          5 OVERPRINT          CHAR (1) INIT ('+');

INCLUDE IDMS (SUBSCHEMA_CTRL);
INCLUDE IDMS (DEPARTMENT);
INCLUDE IDMS (EMPLOYEE);
    /*********************************************************/
    /*    PROCESSING FOLLOWS                                  */
                             /* OPEN THE FILES               */
                             /* INFILE —— INPUT              */
                             /* OUTFILE —— OUTPUT            */
                             /* SYSPRINT —— USED BY IDMS_STATUS */
           OPEN FILE (INFILE);
           OPEN FILE (OUTFILE);
           OPEN FILE (SYSPRINT);
           ON ENDFILE (INFILE) EOF = YES;

    /*    BIND RUN UNIT AND RECORDS EXPLICITLY */
           BIND RUN_UNIT
               NODENAME ('')
               DBNAME ('');

           CALL IDMS_STATUS;
           BIND RECORD (EMPLOYEE);
           CALL IDMS_STATUS;
           BIND RECORD (DEPARTMENT);
           CALL IDMS_STATUS;
           READY;
           CALL IDMS_STATUS;
           READ FILE (INFILE) INTO (DEPT_IN_REC);

        DO WHILE ( EOF);

           DB_END_OF_SET = NO;
           DEPT_ID_0410 = DEPT_ID_IN;
           OBTAIN CALC RECORD (DEPARTMENT);
                                       /* 0326 MEANS      */
                                       /* DEPT NOT FOUND  */
           IF ERROR_STATUS = '0326' THEN CALL NO_DEPT;
           ELSE
             DO;
                IF SET (DEPT_EMPLOYEE) EMPTY THEN CALL NO_EMP;
                ELSE
                  CALL NEW_DEPT;
                  DO UNTIL (DB_END_OF_SET);
                     OBTAIN NEXT RECORD (EMPLOYEE)
                         SET (DEPT_EMPLOYEE);
                     IF ERROR_STATUS = '0307' THEN
                         DB_END_OF_SET = YES;
                     ELSE
                         CALL IDMS_STATUS;
                     IF  DB_END_OF_SET THEN
                       DO;
                       /* MOVE FIELDS TO */
                       /* OUTPUT RECORD  */
                          DEPT_ID_OUT   = DEPT_ID_0410;
```

```
                          EMP_ID_OUT    = EMP_ID_0415;
                          EMP_LNAME_OUT = EMP_LAST_NAME_0415;
                          EMP_FNAME_OUT = EMP_FIRST_NAME_0415;
                          CC            = DOUBLE_SPACE;
                          PRINT_LINE    = SPACES;
                          PRINT_LINE    = PRTREC;
                          CALL PRINT_A_LINE;
                       END;   /*  END PRINTING DO */
                  END;    /* END DO UNTIL */
               END;    /* END 0326 ELSE DO */

           READ FILE (INFILE) INTO (DEPT_IN_REC);
       END;     /* END DO WHILE  EOF */
       CALL END_PROCESSING;
NEW_DEPT: PROC;
         PRINT_LINE = SPACES;       /* NEW PAGE FOR EACH */
         CC = NEW_PAGE;             /* DEPARTMENT        */
         PRINT_LINE = DEPT_HEADER;
         CALL PRINT_A_LINE;

         PRINT_LINE = SPACES;
         CC = DOUBLE_SPACE;
         PRINT_LINE = DEPT_ID_0410;
         CALL PRINT_A_LINE;

         PRINT_LINE = SPACES;
         CC = DOUBLE_SPACE;
         PRINT_LINE = PRTHEAD;
         CALL PRINT_A_LINE;

END NEW_DEPT;

NO_DEPT: PROC;
         PRINT_LINE = SPACES;
         CC = NEW_PAGE;
         PRINT_LINE = DEPT_ID_IN;
         CALL PRINT_A_LINE;
         PRINT_LINE = SPACES;
         CC = DOUBLE_SPACE;
         PRINT_LINE = '** DEPARTMENT SPECIFIED ABOVE NOT FOUND **';
         CALL PRINT_A_LINE;
END NO_DEPT;
NO_EMP: PROC;
         PRINT_LINE = SPACES;
         CC = NEW_PAGE;
         PRINT_LINE = DEPT_ID_IN;
         CALL PRINT_A_LINE;

         PRINT_LINE = SPACES;
         CC = DOUBLE_SPACE;
         PRINT_LINE = DEPT_ID_0410;
         CALL PRINT_A_LINE;

         PRINT_LINE = SPACES;
         CC = DOUBLE_SPACE;
         PRINT_LINE = '** DEPARTMENT SPECIFIED IS EMPTY ***';
         CALL PRINT_A_LINE;
END NO_EMP;
```

```
END_PROCESSING: PROC;
       FINISH;
       CLOSE FILE (INFILE);
       CLOSE FILE (OUTFILE);
       CLOSE FILE (SYSPRINT);
END END_PROCESSING;

PRINT_A_LINE: PROC;
 WRITE FILE (OUTFILE) FROM (PRINT_AREA);
END PRINT_A_LINE;


           INCLUDE IDMS (IDMS_STATUS);

END DEPTRPT;
```

E.3.2 Output from the DML precompiler

The following shows the sample program as output from the DML precompiler.

Since the /*DMLIST*/ option is specified, printed output consists of expanded code as well as diagnostics. This output is in the following format:

- **Heading —** The top of each page of the listing contains the name of the DML precompiler being used (IDMSDMLP), the release number of the processor, the name of the listing (Listing of Messages), the date, the time, and the page number.
- **Input listing and DML precompiler-generated code —** The body of the printout contains the program input listing along with the DML precompiler-generated code, formatted as follows:

Column	Explanation
1	Sequence numbers generated by the DML precompiler
12	Line numbers generated by the DML precompiler
19	Line numbers generated by the user program
26	Text of the PL/I source code including text generated by the DML precompiler

- **Warning and Error Messages —** Diagnostics are imbedded in the input listing and DML precompiler-generated code following the errant lines of source code. For a complete description of DML precompiler error messages, refer to *CA-IDMS Messages and Codes*.

```
IDMSDMLP   15.0         COMPUTER ASSOCIATES INTERNATIONAL, INC. DML PROCESSOR FOR PL/I  DATE      TIME      PAGE
                            - - LISTING OF MESSAGES - -                               09/27/99  13445205    0001

             00001      /*RETRIEVAL*/
             00002      /*DMLIST*/
             00003      /*NO_ACTIVITY_LOG*/
             00004      /*SCHEMA_COMMENTS*/
             00005      DEPTRPT: PROC OPTIONS (MAIN) REORDER;
             00006                                     /* DECLARE SUBSCHEMA AND MODE */
DMLP         00008      DCL    (EMPSS01 SUBSCHEMA, EMPSCHM SCHEMA VERSION 100)
             00009                                         MODE (BATCH) DEBUG;
             00010
             00011                                     /* REQUIRED DECLARATIVES */
             00012      DCL    IDMS ENTRY OPTIONS(INTER,ASM);
             00013      DCL    ABORT ENTRY OPTIONS(INTER,ASM);
```

```
                DCL    ADDR BUILTIN;

                      /* CONSTANTS */
                DCL    DEPT_HEADER        CHAR (11)  INIT ('DEPT REPORT');
                DCL    1 HEAD_LINE,
                        5 HEAD_DEPT_ID     CHAR (9) INIT ('DEPT ID  '),
                        5 HEAD_EMP_ID      CHAR (8) INIT ('EMP ID  '),
                        5 HEAD_LNAME       CHAR (17) INIT ('LAST NAME        '),
                        5 HEAD_FNAME       CHAR (10) INIT ('FIRST NAME');

                DCL    PRTHEAD            CHAR (44) DEFINED HEAD_LINE;

                      /* LOGICAL CONSTANTS */
                DCL    YES                BIT(1)    INIT ('1'B);
                DCL    NO                 BIT(1)    INIT ('0'B);
                DCL    EOF                BIT(1)    INIT ('0'B);

                DCL   1 PROGRAM_FLAGS,
                       5 DB_END_OF_SET     BIT(1)    INIT ('0'B);

                       /* FILE DECLARATIONS */
                DCL    INFILE  FILE RECORD INPUT ENV (F BLKSIZE(80));
                DCL    OUTFILE FILE RECORD OUTPUT ENV (F RECSIZE(133) CTLASA );
                DCL    SYSPRINT FILE PRINT;

                       /*   THE FOLLOWING RECORDS ARE DEFINED THROUGH IDD.       */
                       /*  THE DML PRECOMPILER AUTOMATICALLY CONVERTS HYPHENS  */
                       /*   TO UNDERSCORES.                                     */

DMLP            INCLUDE IDMS (DEPT-IN-REC);
DMLP            INCLUDE IDMS (PRT-OUT-REC);

                       /* REDEFINE PRT_OUT_REC */
                DCL    PRTREC                 CHAR (44) DEFINED PRT_OUT_REC;

                DCL     1 PRINT_AREA,
                          5 CC                CHAR (1),
                          5 PRINT_LINE        CHAR (132);

                DCL     1 SPACES              CHAR (132) INIT ( (132) ' ');

                       /* POSSIBLE VALUES FOR CC                         */
                DCL     1 CONTROL_CHARACTERS,
                          5 NEW_PAGE          CHAR (1) INIT ('1'),
                          5 SINGLE_SPACE      CHAR (1) INIT (' '),
                          5 DOUBLE_SPACE      CHAR (1) INIT ('0'),
                          5 TRIPLE_SPACE      CHAR (1) INIT ('-'),
                          5 OVERPRINT         CHAR (1) INIT ('+');

DMLP            INCLUDE IDMS (SUBSCHEMA_CTRL);
DMLP            INCLUDE IDMS (DEPARTMENT);
DMLP            INCLUDE IDMS (EMPLOYEE);

                    /***********************************************************/
                    /*    PROCESSING FOLLOWS                                    */
                                         /* OPEN THE FILES                 */
                                         /* INFILE -- INPUT                */
                                         /* OUTFILE -- OUTPUT              */
                                         /* SYSPRINT -- USED BY IDMS_STATUS */
                           OPEN FILE (INFILE);
                           OPEN FILE (OUTFILE);
                           OPEN FILE (SYSPRINT);
                           ON ENDFILE (INFILE) EOF = YES;

                    /*    BIND RUN UNIT AND RECORDS EXPLICITLY */
DMLP0001                   BIND RUN_UNIT
                               NODENAME ('')
                               DBNAME ('');

                           CALL IDMS_STATUS;
DMLP0002                   BIND RECORD (EMPLOYEE);
                           CALL IDMS_STATUS;
DMLP0003                   BIND RECORD (DEPARTMENT);
                           CALL IDMS_STATUS;
DMLP0004                   READY;
                           CALL IDMS_STATUS;
                           READ FILE (INFILE) INTO (DEPT_IN_REC);

                        DO WHILE ( EOF);

                           DB_END_OF_SET = NO;
                           DEPT_ID_0410 = DEPT_ID_IN;
DMLP0005                   OBTAIN CALC RECORD (DEPARTMENT);
                                                 /* 0326 MEANS      */
                                                 /* DEPT NOT FOUND  */
                           IF ERROR_STATUS = '0326' THEN CALL NO_DEPT;
                           ELSE
                             DO;
DMLP0006                        IF SET (DEPT_EMPLOYEE) EMPTY
                                                             THEN CALL NO_EMP;
                                ELSE
                                  CALL NEW_DEPT;
                                  DO UNTIL (DB_END_OF_SET);
DMLP0007                             OBTAIN NEXT RECORD (EMPLOYEE)
                                         SET (DEPT_EMPLOYEE);
                                     IF ERROR_STATUS = '0307' THEN
                                         DB_END_OF_SET = YES;
                                     ELSE
                                         CALL IDMS_STATUS;
                                     IF  DB_END_OF_SET THEN
                                       DO;
                                       /* MOVE FIELDS TO */
                                       /* OUTPUT RECORD  */
                                          DEPT_ID_OUT   = DEPT_ID_0410;
                                          EMP_ID_OUT    = EMP_ID_0415;
                                          EMP_LNAME_OUT = EMP_LAST_NAME_0415;
```

```
                                EMP_FNAME_OUT = EMP_FIRST_NAME_0415;
                                CC            = DOUBLE_SPACE;
                                PRINT_LINE    = SPACES;
                                PRINT_LINE    = PRTREC;
                                CALL PRINT_A_LINE;
                             END;   /*  END PRINTING DO */
                        END;    /* END DO UNTIL */
                     END;    /* END 0326 ELSE DO */

                 READ FILE (INFILE) INTO (DEPT_IN_REC);
             END;     /* END DO WHILE  EOF */
             CALL END_PROCESSING;

      NEW_DEPT: PROC;
               PRINT_LINE = SPACES;       /* NEW PAGE FOR EACH */
               CC = NEW_PAGE;             /* DEPARTMENT        */
               PRINT_LINE = DEPT_HEADER;
               CALL PRINT_A_LINE;

               PRINT_LINE = SPACES;
               CC = DOUBLE_SPACE;
               PRINT_LINE = DEPT_ID_0410;
               CALL PRINT_A_LINE;

               PRINT_LINE = SPACES;
               CC = DOUBLE_SPACE;
               PRINT_LINE = PRTHEAD;
               CALL PRINT_A_LINE;

      END NEW_DEPT;

      NO_DEPT: PROC;
               PRINT_LINE = SPACES;
               CC = NEW_PAGE;
               PRINT_LINE = DEPT_ID_IN;
               CALL PRINT_A_LINE;
               PRINT_LINE = SPACES;
               CC = DOUBLE_SPACE;
               PRINT_LINE = '** DEPARTMENT SPECIFIED ABOVE NOT FOUND **';
               CALL PRINT_A_LINE;
      END NO_DEPT;

      NO_EMP: PROC;
               PRINT_LINE = SPACES;
               CC = NEW_PAGE;
               PRINT_LINE = DEPT_ID_IN;
               CALL PRINT_A_LINE;

               PRINT_LINE = SPACES;
               CC = DOUBLE_SPACE;
               PRINT_LINE = DEPT_ID_0410;
               CALL PRINT_A_LINE;

               PRINT_LINE = SPACES;
               CC = DOUBLE_SPACE;
               PRINT_LINE = '** DEPARTMENT SPECIFIED IS EMPTY ***';
               CALL PRINT_A_LINE;
      END NO_EMP;

      END_PROCESSING: PROC;
DMLP0008     FINISH;
             CLOSE FILE (INFILE);
             CLOSE FILE (OUTFILE);
             CLOSE FILE (SYSPRINT);
      END END_PROCESSING;

      PRINT_A_LINE: PROC;
       WRITE FILE (OUTFILE) FROM (PRINT_AREA);
      END PRINT_A_LINE;


DMLP                 INCLUDE IDMS (IDMS_STATUS);
     IDMS_STATUS:  PROC;
         /* THE IDMS_STATUS PROCEDURE IS CALLED BY THE USER AFTER     */
         /* EACH IDMS COMMAND HAS BEEN ISSUED AND CHECKS HAVE BEEN    */
         /* MADE FOR ANY EXPECTED NON-ZERO ERROR_STATUS CONDITIONS.   */
         /* IT DETECTS A NON-ZERO ERROR_STATUS AND ABNORMALLY         */
         /* TERMINATES THE PROGRAM ACCORDINGLY.                       */
         IF   ERROR_STATUS='0000' THEN GOTO END_STATUS;
         PUT SKIP EDIT ('PROGRAM NAME ------', PROGRAM,
                        'ERROR STATUS ------', ERROR_STATUS,
                        'ERROR RECORD ------', ERROR_RECORD,
                        'ERROR SET ---------', ERROR_SET,
                        'ERROR AREA --------', ERROR_AREA,
                        'LAST GOOD RECORD --', RECORD_NAME,
                        'LAST GOOD AREA ----', AREA_NAME,
                        'DML SEQUENCE ------', DML_SEQUENCE)
                       (A(19),X(5),A(8),SKIP,A(19),X(5),A(4),5(SKIP,
                       A(19),X(5),A(16)),SKIP,A(19),X(5),F(10));
DMLP0009 ROLLBACK;
         CALL ABORT;
     END_STATUS: END;

      END DEPTRPT;
```

E.3.3 Output from the PL/I compiler

The following shows the sample batch program after processing by the PL/I compiler. The original code is further expanded and includes the following:

- Line numbers generated by the PL/I compiler
- CA-IDMS call statements for the requested DML functions
- Diagnostic messages

For details on the expanded code generated by the DML precompiler, see Appendix B, “Call Formats” on page B-1.

```
PL/I OPTIMIZING COMPILER         /*RETRIEVAL*/                                                  PAGE   2
                    SOURCE LISTING
    STMT LEV NT

                  /*RETRIEVAL*/
                  /*DMLIST*/
                  /*NO_ACTIVITY_LOG*/
                  /*SCHEMA_COMMENTS*/
       1      0   DEPTRPT: PROC OPTIONS (MAIN) REORDER;
                                              /* DECLARE SUBSCHEMA AND MODE */
                                                                                  /*
                  DCL    (EMPSS01 SUBSCHEMA, EMPSCHM SCHEMA VERSION 100)
                                                  MODE (BATCH) DEBUG;
                                                                                  */

                                              /* REQUIRED DECLARATIVES */
       2   1  0   DCL    IDMS ENTRY OPTIONS(INTER,ASM);
       3   1  0   DCL    ABORT ENTRY OPTIONS(INTER,ASM);
       4   1  0   DCL    ADDR BUILTIN;

                        /* CONSTANTS */
       5   1  0   DCL    DEPT_HEADER         CHAR (11)  INIT ('DEPT REPORT');
       6   1  0   DCL    1 HEAD_LINE,
                          5 HEAD_DEPT_ID     CHAR (9) INIT ('DEPT ID  '),
                          5 HEAD_EMP_ID      CHAR (8) INIT ('EMP ID  '),
                          5 HEAD_LNAME       CHAR (17) INIT ('LAST NAME        '),
                          5 HEAD_FNAME       CHAR (10) INIT ('FIRST NAME');

       7   1  0   DCL    PRTHEAD             CHAR (44) DEFINED HEAD_LINE;

                        /* LOGICAL CONSTANTS */
       8   1  0   DCL    YES                 BIT(1)    INIT ('1'B);
       9   1  0   DCL    NO                  BIT(1)    INIT ('0'B);
      10   1  0   DCL    EOF                 BIT(1)    INIT ('0'B);

      11   1  0   DCL   1 PROGRAM_FLAGS,
                         5 DB_END_OF_SET     BIT(1)    INIT ('0'B);
                         /* FILE DECLARATIONS */
      12   1  0   DCL    INFILE  FILE RECORD INPUT ENV (F BLKSIZE(80));
      13   1  0   DCL    OUTFILE FILE RECORD OUTPUT ENV (F RECSIZE(133) CTLASA );
      14   1  0   DCL    SYSPRINT FILE PRINT;

                         /*   THE FOLLOWING RECORDS ARE DEFINED THROUGH IDD.       */
                         /*  THE DML PRECOMPILER AUTOMATICALLY CONVERTS HYPHENS  */
                         /*   TO UNDERSCORES.                                   */

                                                                                  /*
                  INCLUDE IDMS (DEPT-IN-REC);
      15   1  0  DECLARE 1  DEPT_IN_REC,
                          2  DEPT_ID_IN PICTURE '(4)9',
                          2  DEPT_FILLER CHARACTER (76);
                                                                                  /*
                  INCLUDE IDMS (PRT-OUT-REC);
                                                                                  */
      16   1  0  DECLARE 1  PRT_OUT_REC,
                          2  DEPT_ID_OUT CHARACTER (4),
                          2  PRT_FILL_5 CHARACTER (5) INITIAL  ('     '),
                          2  EMP_ID_OUT CHARACTER (4),
                          2  PRT_FILL_4 CHARACTER (4) INITIAL  ('    '),
                          2  EMP_LNAME_OUT CHARACTER (15),
                          2  PRT_FILL_2 CHARACTER (2) INITIAL  ('  '),
                          2  EMP_FNAME_OUT CHARACTER (10);

                         /* REDEFINE PRT_OUT_REC */
      17   1  0   DCL    PRTREC                  CHAR (44) DEFINED PRT_OUT_REC;

      18   1  0   DCL     1 PRINT_AREA,
                            5 CC                  CHAR (1),
                            5 PRINT_LINE          CHAR (132);

      19   1  0   DCL     1 SPACES                CHAR (132) INIT ( (132) ' ');

                         /* POSSIBLE VALUES FOR CC                         */
      20   1  0   DCL     1 CONTROL_CHARACTERS,
                            5 NEW_PAGE            CHAR (1) INIT ('1'),
                            5 SINGLE_SPACE        CHAR (1) INIT (' '),
                            5 DOUBLE_SPACE        CHAR (1) INIT ('0'),
                            5 TRIPLE_SPACE        CHAR (1) INIT ('-'),
                            5 OVERPRINT           CHAR (1) INIT ('+');
                                                                                  /*
```

```
           INCLUDE IDMS (SUBSCHEMA_CTRL);
                                                                  *
21  DECLARE 1  SUBSCHEMA_CTRL,
              3  PROGRAM CHARACTER (8) INITIAL  (' ') ,
              3  ERROR_STATUS CHARACTER (4) INITIAL  ('1400') ,
              3  DBKEY FIXED BINARY (31),
              3  RECORD_NAME CHARACTER (16) INITIAL  (' ') ,
              3  AREA_NAME CHARACTER (16) INITIAL  (' ') ,
              3  ERROR_SET CHARACTER (16) INITIAL  (' ') ,
              3  ERROR_RECORD CHARACTER (16) INITIAL  (' ') ,
              3  ERROR_AREA CHARACTER (16) INITIAL  (' ') ,
              3  IDBMSCOM_AREA CHARACTER (100) INITIAL  (LOW(100)) ,
              3  DIRECT_DBKEY FIXED BINARY (31),
              3  DATABASE_STATUS,
                5  DBSTATMENT_CODE CHARACTER (2),
                5  DBSTATUS_CODE CHARACTER (5),
              3  FILLER0001 CHARACTER (1),
              3  RECORD_OCCUR FIXED BINARY (31),
              3  DML_SEQUENCE FIXED BINARY (31);
22  DECLARE 1  RIDBMSCOM BASED(ADDR(SUBSCHEMA_CTRL.IDBMSCOM_AREA)),
              3  PAGE_INFO,
                5  PAGE_INFO_GROUP FIXED BINARY (15),
                5  PAGE_INFO_DBK_FORMAT FIXED BINARY (15),
              3  FILLER0002 CHARACTER (96);
23  DECLARE 1  IDBMSCOM (100) BASED(ADDR(SUBSCHEMA_CTRL.IDBMSCOM_AREA))
                                         CHARACTER (1);
24  DECLARE 1  AREA_RNAME BASED(ADDR(SUBSCHEMA_CTRL.AREA_NAME)),
              3  SSC_DNO CHARACTER (8),
              3  SSC_DNA CHARACTER (8);
25  DECLARE 1  RRECORD_NAME BASED(ADDR(SUBSCHEMA_CTRL.RECORD_NAME)),
              3  SSC_NODN CHARACTER (8),
              3  SSC_DBN CHARACTER (8);
26   1  0  DECLARE 1  SUBSCHEMA_CTRL,
                     3  PROGRAM CHARACTER (8) INITIAL  (' '),
                     3  ERROR_STATUS CHARACTER (4) INITIAL  ('1400'),
                     3  DBKEY FIXED BINARY (31),
                     3  RECORD_NAME CHARACTER (16) INITIAL  (' '),
                     3  AREA_NAME CHARACTER (16) INITIAL  (' '),
                     3  ERROR_SET CHARACTER (16) INITIAL  (' '),
                     3  ERROR_RECORD CHARACTER (16) INITIAL  (' '),
                     3  ERROR_AREA CHARACTER (16) INITIAL  (' '),
                     3  IDBMSCOM_AREA,
                       5  IDBMSCOM (100) CHARACTER (1),
                     3  DIRECT_DBKEY FIXED BINARY (31),
                     3  DATABASE_STATUS,
                       5  DBSTATMENT_CODE CHARACTER (2),
                       5  DBSTATUS_CODE CHARACTER (5),
                     3  FILLER0001 CHARACTER (1),
                     3  RECORD_OCCUR FIXED BINARY (31),
                     3  DML_SEQUENCE FIXED BINARY (31);
27   1  0  DECLARE 1  AREA_RNAME BASED(ADDR(SUBSCHEMA_CTRL.AREA_NAME)),
                     3  SSC_DNO CHARACTER (8),
                     3  SSC_DNA CHARACTER (8);
28   1  0  DECLARE 1  RRECORD_NAME BASED(ADDR(SUBSCHEMA_CTRL.RECORD_NAME)),
                     3  SSC_NODN CHARACTER (8),
                     3  SSC_DBN CHARACTER (8);
                                                                         /*
           INCLUDE IDMS (DEPARTMENT);
                                                                         */
28   1  0  DECLARE 1  DEPARTMENT,
                    2  DEPT_ID_0410 PICTURE '(4)9',
                    2  DEPT_NAME_0410 CHARACTER (45),
                    2  DEPT_HEAD_ID_0410 PICTURE '(4)9',
                    2  FILLER0002 CHARACTER (3);
                                                                         /*
           INCLUDE IDMS (EMPLOYEE);
                                                                         */
30   1  0  DECLARE 1  EMPLOYEE,
                    2  EMP_ID_0415 PICTURE '(4)9',
                    2  EMP_NAME_0415,
                     3  EMP_FIRST_NAME_0415 CHARACTER (10),
                     3  EMP_LAST_NAME_0415 CHARACTER (15),
                    2  EMP_ADDRESS_0415,
                     3  EMP_STREET_0415 CHARACTER (20),
                     3  EMP_CITY_0415 CHARACTER (15),
                     3  EMP_STATE_0415 CHARACTER (2),
                     3  EMP_ZIP_0415,
                      4  EMP_ZIP_FIRST_FIVE_0415 CHARACTER (5),
                      4  EMP_ZIP_LAST_FOUR_0415 CHARACTER (4),
                    2  EMP_PHONE_0415 PICTURE '(10)9',
                    2  STATUS_0415 CHARACTER (2),
                    2  SS_NUMBER_0415 PICTURE '(9)9',
                    2  START_DATE_0415,
                     3  START_YEAR_0415 PICTURE '(2)9',
                     3  START_MONTH_0415 PICTURE '(2)9',
                     3  START_DAY_0415 PICTURE '(2)9',
                    2  TERMINATION_DATE_0415,
                     3  TERMINATION_YEAR_0415 PICTURE '(2)9',
                     3  TERMINATION_MONTH_0415 PICTURE '(2)9',
                     3  TERMINATION_DAY_0415 PICTURE '(2)9',
                    2  BIRTH_DATE_0415,
                     3  BIRTH_YEAR_0415 PICTURE '(2)9',
                     3  BIRTH_MONTH_0415 PICTURE '(2)9',
                     3  BIRTH_DAY_0415 PICTURE '(2)9',
                    2  FILLER0003 CHARACTER (2),
                    2  FILLER0004 CHARACTER (4);
                /*********************************************************/
                /*    PROCESSING FOLLOWS                                  */
                                         /* OPEN THE FILES                */
                                         /* INFILE –– INPUT               */
                                         /* OUTFILE –– OUTPUT             */
                                         /* SYSPRINT –– USED BY IDMS_STATUS */
31   1  0              OPEN FILE (INFILE);
32   1  0              OPEN FILE (OUTFILE);
33   1  0              OPEN FILE (SYSPRINT);
34   1  0              ON ENDFILE (INFILE) EOF = YES;
```

```
                 /*    BIND RUN UNIT AND RECORDS EXPLICITLY */
                                                                            /*
                       BIND RUN_UNIT                                          DMLP0001
                           NODENAME ('')
                           DBNAME ('');
                                                                            */
35   1  0                           /* IDMS PL/I DML EXPANSION */           DO;
36   1  1                           DML_SEQUENCE=1;
37   1  1                           SSC_NODN='';
38   1  1                           SSC_DBN='';
39   1  1                           CALL IDMS (SUBSCHEMA_CTRL
                                                      ,IDBMSCOM (59)
                                                      ,SUBSCHEMA_CTRL
                                                      ,'EMPSS01 '
40   1  1                                                               ); END;

41   1  0                  CALL IDMS_STATUS;
                                                                            /*
                           BIND RECORD (EMPLOYEE);                            DMLP0002
                                                                            */
42   1  0                           /* IDMS PL/I DML EXPANSION */           DO;
43   1  1                           DML_SEQUENCE=2;
44   1  1                           CALL IDMS (SUBSCHEMA_CTRL
                                                      ,IDBMSCOM (48)
                                                      ,'EMPLOYEE        '
                                                      ,EMPLOYEE
45   1  1                                                               ); END;
46   1  0                  CALL IDMS_STATUS;
                                                                            /*
                           BIND RECORD (DEPARTMENT);                          DMLP0003
                                                                            */
47   1  0                           /* IDMS PL/I DML EXPANSION */           DO;
48   1  1                           DML_SEQUENCE=3;
49   1  1                           CALL IDMS (SUBSCHEMA_CTRL
                                                      ,IDBMSCOM (48)
                                                      ,'DEPARTMENT      '
                                                      ,DEPARTMENT
50   1  1                                                               ); END;
51   1  0                  CALL IDMS_STATUS;
                                                                            /*
                           READY;                                             DMLP0004
                                                                            */
52   1  0                           /* IDMS PL/I DML EXPANSION */           DO;
53   1  1                           DML_SEQUENCE=4;
54   1  1                           CALL IDMS (SUBSCHEMA_CTRL
                                                      ,IDBMSCOM (37)
55   1  1                                                               ); END;
56   1  0                  CALL IDMS_STATUS;
57   1  0                  READ FILE (INFILE) INTO (DEPT_IN_REC);

58   1  0               DO WHILE ( EOF);

59   1  1                  DB_END_OF_SET = NO;
60   1  1                  DEPT_ID_0410 = DEPT_ID_IN;
                                                                            /*
                           OBTAIN CALC RECORD (DEPARTMENT);                   DMLP0005
                                                                            */
61   1  1                           /* IDMS PL/I DML EXPANSION */           DO;
62   1  2                           DML_SEQUENCE=5;
63   1  2                           CALL IDMS (SUBSCHEMA_CTRL
                                                      ,IDBMSCOM (32)
                                                      ,'DEPARTMENT      '
                                                      ,IDBMSCOM (43)
64   1  2                                                               ); END;
                                                       /* 0326 MEANS      */
                                                       /* DEPT NOT FOUND  */
65   1  1                  IF ERROR_STATUS = '0326' THEN CALL NO_DEPT;
66   1  1                  ELSE
                             DO;
                                                                           /*
                                IF SET (DEPT_EMPLOYEE) EMPTY                  DMLP0006
                                                                            */
67   1  2                           /* IDMS PL/I DML EXPANSION */           DO;
68   1  3                           DML_SEQUENCE=6;
69   1  3                           CALL IDMS (SUBSCHEMA_CTRL
                                                      ,IDBMSCOM (64)
                                                      ,'DEPT-EMPLOYEE   '
70   1  3                                                               ); END;
71   1  2                           IF ERROR_STATUS='0000'
                                                            THEN CALL NO_EMP;
72   1  2                       ELSE
                                  CALL NEW_DEPT;
73   1  2                         DO UNTIL (DB_END_OF_SET);
                                                                            /*
                                     OBTAIN NEXT RECORD (EMPLOYEE)            DMLP0007
                                         SET (DEPT_EMPLOYEE);
                                                                            */
74   1  3                           /* IDMS PL/I DML EXPANSION */           DO;
75   1  4                           DML_SEQUENCE=7;
76   1  4                           CALL IDMS (SUBSCHEMA_CTRL
                                                      ,IDBMSCOM (10)
                                                      ,'EMPLOYEE        '
                                                      ,'DEPT-EMPLOYEE   '
                                                      ,IDBMSCOM (43)
77   1  4                                                               ); END;
78   1  3                            IF ERROR_STATUS = '0307' THEN
                                         DB_END_OF_SET = YES;
79   1  3                            ELSE
                                         CALL IDMS_STATUS;
80   1  3                            IF  DB_END_OF_SET THEN
                                       DO;
                                       /* MOVE FIELDS TO */
                                       /* OUTPUT RECORD  */
81   1  4                                 DEPT_ID_OUT   = DEPT_ID_0410;
82   1  4                                 EMP_ID_OUT    = EMP_ID_0415;
```

```
 83   1  4                            EMP_LNAME_OUT = EMP_LAST_NAME_0415;
 84   1  4                            EMP_FNAME_OUT = EMP_FIRST_NAME_0415;
 85   1  4                            CC            = DOUBLE_SPACE;
 86   1  4                            PRINT_LINE    = SPACES;
 87   1  4                            PRINT_LINE    = PRTREC;
 88   1  4                            CALL PRINT_A_LINE;
 89   1  4                         END;   /*  END PRINTING DO */
 90   1  3                    END;    /* END DO UNTIL */
 91   1  2                 END;    /* END 0326 ELSE DO */

 92   1  1             READ FILE (INFILE) INTO (DEPT_IN_REC);
 93   1  1         END;     /* END DO WHILE  EOF */
 94   1  0         CALL END_PROCESSING;

 95   1  0   NEW_DEPT: PROC;
 96   2  0           PRINT_LINE = SPACES;       /* NEW PAGE FOR EACH */
 97   2  0           CC = NEW_PAGE;             /* DEPARTMENT        */
 98   2  0           PRINT_LINE = DEPT_HEADER;
 99   2  0           CALL PRINT_A_LINE;

100   2  0           PRINT_LINE = SPACES;
101   2  0           CC = DOUBLE_SPACE;
102   2  0           PRINT_LINE = DEPT_ID_0410;
103   2  0           CALL PRINT_A_LINE;

104   2  0           PRINT_LINE = SPACES;
105   2  0           CC = DOUBLE_SPACE;
106   2  0           PRINT_LINE = PRTHEAD;
107   2  0           CALL PRINT_A_LINE;

108   2  0   END NEW_DEPT;
109   1  0   NO_DEPT: PROC;
110   2  0           PRINT_LINE = SPACES;
111   2  0           CC = NEW_PAGE;
112   2  0           PRINT_LINE = DEPT_ID_IN;
113   2  0           CALL PRINT_A_LINE;
114   2  0           PRINT_LINE = SPACES;
115   2  0           CC = DOUBLE_SPACE;
116   2  0           PRINT_LINE = '** DEPARTMENT SPECIFIED ABOVE NOT FOUND **';
117   2  0           CALL PRINT_A_LINE;
118   2  0   END NO_DEPT;

119   1  0   NO_EMP: PROC;
120   2  0           PRINT_LINE = SPACES;
121   2  0           CC = NEW_PAGE;
122   2  0           PRINT_LINE = DEPT_ID_IN;
123   2  0           CALL PRINT_A_LINE;

124   2  0           PRINT_LINE = SPACES;
125   2  0           CC = DOUBLE_SPACE;
126   2  0           PRINT_LINE = DEPT_ID_0410;
127   2  0           CALL PRINT_A_LINE;

128   2  0           PRINT_LINE = SPACES;
129   2  0           CC = DOUBLE_SPACE;
130   2  0           PRINT_LINE = '** DEPARTMENT SPECIFIED IS EMPTY ***';
131   2  0           CALL PRINT_A_LINE;
132   2  0   END NO_EMP;

133   1  0   END_PROCESSING: PROC;
                                                                              /*
                 FINISH;                                                        DMLP0008
                                                                              */
134   2  0                      /* IDMS PL/I DML EXPANSION */           DO;
135   2  1                      DML_SEQUENCE=8;
136   2  1                      CALL IDMS (SUBSCHEMA_CTRL
                                                 ,IDBMSCOM (2)
137   2  1                                                         ); END;
138   2  0         CLOSE FILE (INFILE);
139   2  0         CLOSE FILE (OUTFILE);
140   2  0         CLOSE FILE (SYSPRINT);
141   2  0   END END_PROCESSING;

142   1  0   PRINT_A_LINE: PROC;
143   2  0    WRITE FILE (OUTFILE) FROM (PRINT_AREA);
144   2  0   END PRINT_A_LINE;

                    INCLUDE IDMS (IDMS_STATUS);
                                                                              */
145   1  0  IDMS_STATUS:  PROC;
               /* THE IDMS_STATUS PROCEDURE IS CALLED BY THE USER AFTER     */
               /* EACH IDMS COMMAND HAS BEEN ISSUED AND CHECKS HAVE BEEN    */
               /* MADE FOR ANY EXPECTED NON-ZERO ERROR_STATUS CONDITIONS.   */
               /* IT DETECTS A NON-ZERO ERROR_STATUS AND ABNORMALLY         */
               /* TERMINATES THE PROGRAM ACCORDINGLY.                       */
146   2  0     IF   ERROR_STATUS='0000' THEN GOTO END_STATUS;
147   2  0     PUT SKIP EDIT ('PROGRAM NAME ------', PROGRAM,
                             'ERROR STATUS ------', ERROR_STATUS,
                             'ERROR RECORD ------', ERROR_RECORD,
                             'ERROR SET ---------', ERROR_SET,
                             'ERROR AREA --------', ERROR_AREA,
                             'LAST GOOD RECORD --', RECORD_NAME,
                             'LAST GOOD AREA ----', AREA_NAME,
                             'DML SEQUENCE ------', DML_SEQUENCE)
                            (A(19),X(5),A(8),SKIP,A(19),X(5),A(4),5(SKIP,
                            A(19),X(5),A(16)),SKIP,A(19),X(5),F(10));
                                                                              /*
               ROLLBACK;
                                                                              */
148   2  0                      /* IDMS PL/I DML EXPANSION */           DO;
149   2  1                      DML_SEQUENCE=9;
150   2  1                      CALL IDMS (SUBSCHEMA_CTRL
                                                 ,IDBMSCOM (67)
151   2  1                                                         ); END;
152   2  0     CALL ABORT;
153   2  0  END_STATUS: END;
```

```
154   1  0   END DEPTRPT;
```

E.4 Sample online program

The following CA-IDMS/DC application illustrates the structure of CA-IDMS/DC programs that accept data from a terminal operator and retrieve information from the database. The application program highlights the following database and data communications features:

- Mapping mode input and output
- Automatic editing and error handling
- Pseudo-conversational transactions

The application's components, runtime requirements, and DML code are described in the following subsections.

E.4.1 Application components

The application comprises a program, two tasks, a map, and a subschema:

- **Program** — The EMPDISP program either performs a MAP OUT to start a session or performs a MAP IN, database access, and a MAP OUT.
- **Tasks —** The task codes EMPDISP and EMPDISP2 affect the program flow of control:
  - **EMPDISP** causes the program to perform the FIRST_TIME portion of the program, mapping out the empty screen.
  - **EMPDISP2** causes the program to perform the SECOND_TIME portion of the program, mapping in the data, checking the AID byte, performing the database access portion of the program, and mapping out either an error message or employee data.
-

Map — The application uses a map named EMPLMAP to communicate with the terminal operator. The following illustrates the EMPLMAP map.

```
                         *** EMPLOYEE INFORMATION SCREEN ***

                EMPLOYEE ID:

                FIRST NAME:
                LAST NAME :

                ADDRESS:
                       :
                       :        :








            TYPE AN EMPLOYEE ID AND PRESS ENTER ** PRESS PA1 TO EXIT
```

The EMPLMAP definition specifies:

- Six literal fields (including the title EMPLOYEE INFORMATION SCREEN).
- Seven variable data fields, to contain: EMPLOYEE ID, LAST NAME, FIRST NAME, and ADDRESS.
- Automatic editing for the EMPLOYEE ID field specifies that the field is in error if the ID you entered does not comply with the field's external picture (PIC 9(4)).
- Messages are output in the $MESSAGE field.

- **Subschema** — The application uses the EMPSS01 subschema.

E.4.2 Application runtime requirements

The following requirements must be met to execute the sample application under CA-IDMS/DC:

- Define and generate the EMPLMAP map.
- Compile and link edit the EMPDISP program into a load library that is identified to CA-IDMS/DC.
- Define the EMPDISP program to the CA-IDMS/DC system either by submitting PROGRAM statements to the system generation compiler or by using the DCMT VARY DYNAMIC PROGRAM command at runtime.
- Define the EMPLMAP map and the EMPSS01 subschema to the CA-IDMS/DC system by submitting PROGRAM statements to the system generation compiler.

Maps and subschemas are defined automatically at system startup if null program definition elements (PDEs) have been allocated for them at system generation.

E.4.3 Online input to the DML precompiler

The following is the PL/I online program input to the DML precompiler.

```
/*RETRIEVAL*/
/*DMLIST*/
/*NO_ACTIVITY_LOG*/
/*SCHEMA_COMMENTS*/
EMPDISP: PROC OPTIONS (MAIN) REORDER;
DCL    (EMPSS01 SUBSCHEMA, EMPSCHM SCHEMA VERSION 100)
                                   MODE (IDMS_DC) DEBUG;
DCL    IDMSPLI ENTRY OPTIONS(INTER,ASM);
DCL    ADDR BUILTIN;
DCL    STRING BUILTIN;
DCL    (EMPLMAP MAP) TYPE (STANDARD);

DCL    TASK_CODE            CHAR (8);
DCL    EMPDISP              CHAR (8) INIT ('EMPDISP');
DCL    EMPDISP2             CHAR (8) INIT ('EMPDISP2');
DCL    DC_AID_IND_V         CHAR (1);
       /* LOGICAL CONSTANTS */
DCL    YES                  BIT(1)    INIT ('1'B);
DCL    NO                   BIT(1)    INIT ('0'B);
DCL   1 PROGRAM_MESSAGES,
          3 DISPLAY_MSG     CHAR (36)
            INIT (' EMPLOYEE INFORMATION DISPLAYED '),
          3 NOT_FOUND_MSG        CHAR (37)
            INIT (' SPECIFIED EMPLOYEE NUMBER NOT FOUND ');

INCLUDE IDMS (SUBSCHEMA_CTRL);

INCLUDE IDMS (EMPLOYEE);
INCLUDE IDMS (MAP_CONTROLS);
/*        PROCESSING FOLLOWS                         */

MAIN_LINE: BEGIN;
                                 /* ESTABLISH ADDRESSABILITY FOR */
           BIND MAP (EMPLMAP);
           CALL IDMS_STATUS;
           BIND MAP (EMPLMAP) RECORD (EMPLOYEE);
           CALL IDMS_STATUS;
                                  /* DETERMINE THE TASK CODE */
           ACCEPT TASK CODE INTO (TASK_CODE);
           CALL IDMS_STATUS;

           IF TASK_CODE = EMPDISP
           THEN CALL FIRST_TIME;
           IF TASK_CODE = EMPDISP2
           THEN CALL SECOND_TIME;



                          /* OTHERWISE RETURN TO IDMS-DC */
           DC RETURN;

FIRST_TIME: PROC;
  MODIFY MAP (EMPLMAP)
     FOR ALL BUT DFLD (EMP_ID_0415)
```

```
     ATTRIBUTES PROTECTED;

  MAP OUT(EMPLMAP)
     IO OUTPUT DATA YES NEWPAGE;
  CALL IDMS_STATUS;
  DC RETURN NEXT TASK CODE(EMPDISP2);
END FIRST_TIME;

SECOND_TIME: PROC;
   MAP IN (EMPLMAP)
      IO INPUT DATA YES;
   CALL IDMS_STATUS;
                             /* CHECK WHICH PF KEY WAS PRESSED */
   INQUIRE MAP(EMPLMAP)
       MOVE AID TO (DC_AID_IND_V);

                             /* STOP IF PA1 (%) WAS PRESSED */
   IF DC_AID_IND_V = '%'
       THEN DC RETURN;
   BIND RUN_UNIT;
   CALL IDMS_STATUS;
   BIND RECORD (EMPLOYEE);
   CALL IDMS_STATUS;
   READY AREA (EMP_DEMO_REGION);
   CALL IDMS_STATUS;
                           /* OBTAIN THE RECORD */
   OBTAIN CALC RECORD (EMPLOYEE);
   IF ERROR_STATUS = '0326' THEN CALL NO_EMP;
   CALL IDMS_STATUS;
   FINISH;
   CALL IDMS_STATUS;
                    /* TRANSMIT THE DATA BACK TO THE SCREEN */
   MAP OUT(EMPLMAP)
      IO OUTPUT DATA YES NEWPAGE
      MESSAGE(DISPLAY_MSG) LENGTH(36);
   CALL IDMS_STATUS;
   DC RETURN NEXT TASK CODE(EMPDISP2);

END SECOND_TIME;

NO_EMP: PROC;
                    /* DO THIS IF EMPLOYEE NOT FOUND */
  MAP OUT(EMPLMAP)
     IO OUTPUT DATA YES NEWPAGE
     MESSAGE(NOT_FOUND_MSG) LENGTH(37);
  CALL IDMS_STATUS;
  DC RETURN NEXT TASK CODE(EMPDISP2);
END NO_EMP;

           INCLUDE IDMS (IDMS_STATUS);
END MAIN_LINE; /* END MAIN_LINE */
END EMPDISP;
```

E.4.4 Output from the DML precompiler

The following is the online program as it has been output from the DML precompiler.

```
IDMSDMLP   15.0         COMPUTER ASSOCIATES INTERNATIONAL, INC. DML PROCESSOR FOR PL/I  DATE      TIME      PAGE
                            - - LISTING OF MESSAGES - -                            09/27/99  13395209    0001

             00001      /*RETRIEVAL*/
             00002      /*DMLIST*/
             00003      /*NO_ACTIVITY_LOG*/
             00004      /*SCHEMA_COMMENTS*/
             00005      EMPDISP: PROC OPTIONS (MAIN) REORDER;
DMLP         00007      DCL    (EMPSS01 SUBSCHEMA, EMPSCHM SCHEMA VERSION 100)
             00008                                       MODE (IDMS_DC) DEBUG;
             00009      DCL    IDMSPLI ENTRY OPTIONS(INTER,ASM);
             00010      DCL    ADDR BUILTIN;
             00011      DCL    STRING BUILTIN;
DMLP         00013      DCL    (EMPLMAP MAP) TYPE (STANDARD);
             00014
             00015      DCL    TASK_CODE           CHAR (8);
             00016      DCL    EMPDISP             CHAR (8) INIT ('EMPDISP');
             00017      DCL    EMPDISP2            CHAR (8) INIT ('EMPDISP2');
             00018      DCL    DC_AID_IND_V        CHAR (1);
             00019             /* LOGICAL CONSTANTS */
             00020      DCL    YES                 BIT(1)    INIT ('1'B);
             00021      DCL    NO                  BIT(1)    INIT ('0'B);
             00022      DCL   1 PROGRAM_MESSAGES,
             00023               3 DISPLAY_MSG     CHAR (36)
             00024                 INIT (' EMPLOYEE INFORMATION DISPLAYED '),
             00025               3 NOT_FOUND_MSG        CHAR (37)
             00026                 INIT (' SPECIFIED EMPLOYEE NUMBER NOT FOUND ');
             00027
DMLP         00029      INCLUDE IDMS (SUBSCHEMA_CTRL);
             00100
DMLP         00102      INCLUDE IDMS (EMPLOYEE);
DMLP         00133      INCLUDE IDMS (MAP_CONTROLS);
             00171
             00172      /*        PROCESSING FOLLOWS                         */
             00173
             00174      MAIN_LINE: BEGIN;
             00175                                      /* ESTABLISH ADDRESSABILITY FOR */
DMLP0001     00177                 BIND MAP (EMPLMAP);
             00208                 CALL IDMS_STATUS;
DMLP0002     00210                 BIND MAP (EMPLMAP) RECORD (EMPLOYEE);
             00219                 CALL IDMS_STATUS;
             00220                                       /* DETERMINE THE TASK CODE */
DMLP0003     00222                 ACCEPT TASK CODE INTO (TASK_CODE);
             00231                 CALL IDMS_STATUS;
             00232
             00233                 IF TASK_CODE = EMPDISP
             00234                 THEN CALL FIRST_TIME;
             00235                 IF TASK_CODE = EMPDISP2
             00236                 THEN CALL SECOND_TIME;
             00237
             00238
             00239
             00240                               /* OTHERWISE RETURN TO IDMS-DC */
DMLP0004     00242                 DC RETURN;
             00249
             00250      FIRST_TIME: PROC;
DMLP0005     00252        MODIFY MAP (EMPLMAP)
             00253           FOR ALL BUT DFLD (EMP_ID_0415)
             00254           ATTRIBUTES PROTECTED;
             00267
DMLP0006     00269        MAP OUT(EMPLMAP)
             00270           IO OUTPUT DATA YES NEWPAGE;
             00284        CALL IDMS_STATUS;
DMLP0007     00286        DC RETURN NEXT TASK CODE(EMPDISP2);
             00295      END FIRST_TIME;
             00296
             00297      SECOND_TIME: PROC;
DMLP0008     00299         MAP IN (EMPLMAP)
             00300            IO INPUT DATA YES;
             00314         CALL IDMS_STATUS;
             00315                                     /* CHECK WHICH PF KEY WAS PRESSED */
DMLP0009     00317         INQUIRE MAP(EMPLMAP)
             00318             MOVE AID TO (DC_AID_IND_V);
             00328
             00329                                     /* STOP IF PA1 (%) WAS PRESSED */
             00330         IF DC_AID_IND_V = '%'
DMLP0010     00331             THEN
             00333                  DC RETURN;
             00340
DMLP0011     00342         BIND RUN_UNIT;
             00351         CALL IDMS_STATUS;
DMLP0012     00353         BIND RECORD (EMPLOYEE);
             00362         CALL IDMS_STATUS;
DMLP0013     00364         READY AREA (EMP_DEMO_REGION);
             00372         CALL IDMS_STATUS;
             00373                                   /* OBTAIN THE RECORD */
DMLP0014     00375         OBTAIN CALC RECORD (EMPLOYEE);
             00384         IF ERROR_STATUS = '0326' THEN CALL NO_EMP;
             00385         CALL IDMS_STATUS;
DMLP0015     00387         FINISH;
             00394         CALL IDMS_STATUS;
             00395                          /* TRANSMIT THE DATA BACK TO THE SCREEN */
DMLP0016     00397         MAP OUT(EMPLMAP)
             00398            IO OUTPUT DATA YES NEWPAGE
             00399            MESSAGE(DISPLAY_MSG) LENGTH(36);
             00415         CALL IDMS_STATUS;
DMLP0017     00417         DC RETURN NEXT TASK CODE(EMPDISP2);
             00426
```

```
          00427       END SECOND_TIME;
          00428
          00429       NO_EMP: PROC;
          00430                             /* DO THIS IF EMPLOYEE NOT FOUND */
DMLP0018  00432         MAP OUT(EMPLMAP)
          00433            IO OUTPUT DATA YES NEWPAGE
          00434            MESSAGE(NOT_FOUND_MSG) LENGTH(37);
          00450         CALL IDMS_STATUS;
DMLP0019  00452         DC RETURN NEXT TASK CODE(EMPDISP2);
          00461       END NO_EMP;
          00462
DMLP      00464                 INCLUDE IDMS (IDMS_STATUS);
          00465      IDMS_STATUS:  PROC;
          00466      /* THE IDMS_STATUS PROCEDURE MAY BE CALLED BY THE USER AFTER */
          00467      /* EACH IDMS COMMAND HAS BEEN ISSUED AND CHECKS HAVE BEEN    */
          00468      /* MADE FOR ANY EXPECTED NON_ZERO ERROR STATUS CONDITIONS.   */
          00469      /* IT DETECTS A NON_ZERO ERROR_STATUS AND TERMINATES THE     */
          00470      /* PROGRAM WITH A SNAP OF THE SUBSCHEMA_CTRL AREA AND AN     */
          00471      /* ABEND WITH THE ERROR_STATUS AS THE ABEND CODE.            */
          00472          IF ERROR_STATUS='0000' THEN GOTO END_STATUS;
          00473          SSC_ERRSTAT_SAVE=ERROR_STATUS; /* SAVE THE ERROR_STATUS */
          00474          SSC_DMLSEQ_SAVE=DML_SEQUENCE; /* SAVE DML_SEQUENCE     */
          00475      /* SNAP THE SUBSCHEMA_CTRL AREA */
DMLP0020  00477          SNAP FROM (SUBSCHEMA_CTRL) TO (SUBSCHEMA_CTRL_END);
          00490      /* ABEND */
DMLP0021  00492          ABEND CODE (SSC_ERRSTAT_SAVE);
          00501      END_STATUS: END;
          00502       END MAIN_LINE; /* END MAIN_LINE */
          00503       END EMPDISP;
```

E.4.5 Output from the PL/I compiler

The following is the PL/I program as output by the PL/I compiler.

```
PL/I OPTIMIZING COMPILER         /*RETRIEVAL*/                                                                  PAGE   2
                    SOURCE LISTING
    STMT LEV NT

                  /*RETRIEVAL*/
                  /*DMLIST*/
                  /*NO_ACTIVITY_LOG*/
                  /*SCHEMA_COMMENTS*/
       1      0   EMPDISP: PROC OPTIONS (MAIN) REORDER;
                                                                                           /*
                  DCL    (EMPSS01 SUBSCHEMA, EMPSCHM SCHEMA VERSION 100)
                                                    MODE (IDMS_DC) DEBUG;
                                                                                           */
       2   1  0   DCL    IDMSPLI ENTRY OPTIONS(INTER,ASM);
       3   1  0   DCL    ADDR BUILTIN;
       4   1  0   DCL    STRING BUILTIN;
                                                                                           /*
                  DCL    (EMPLMAP MAP) TYPE (STANDARD);
                                                                                           */

       5   1  0   DCL    TASK_CODE           CHAR (8);
       6   1  0   DCL    EMPDISP             CHAR (8) INIT ('EMPDISP');
       7   1  0   DCL    EMPDISP2            CHAR (8) INIT ('EMPDISP2');
       8   1  0   DCL    DC_AID_IND_V        CHAR (1);
                         /* LOGICAL CONSTANTS */
       9   1  0   DCL    YES                 BIT(1)    INIT ('1'B);
      10   1  0   DCL    NO                  BIT(1)    INIT ('0'B);
      11   1  0   DCL   1 PROGRAM_MESSAGES,
                            3 DISPLAY_MSG     CHAR (36)
                              INIT (' EMPLOYEE INFORMATION DISPLAYED '),
                            3 NOT_FOUND_MSG        CHAR (37)
                              INIT (' SPECIFIED EMPLOYEE NUMBER NOT FOUND ');

                                                                                           /*
                  INCLUDE IDMS (SUBSCHEMA_CTRL);
                                                                                           */
      12   1  0  DECLARE 1  SUBSCHEMA_CTRL,
                           3  PROGRAM CHARACTER (8) INITIAL  (' '),
                           3  ERROR_STATUS CHARACTER (4) INITIAL  ('1400'),
                           3  DBKEY FIXED BINARY (31),
                           3  RECORD_NAME CHARACTER (16) INITIAL  (' '),
                           3  AREA_NAME CHARACTER (16) INITIAL  (' '),
                           3  ERROR_SET CHARACTER (16) INITIAL  (' '),
                           3  ERROR_RECORD CHARACTER (16) INITIAL  (' '),
                           3  ERROR_AREA CHARACTER (16) INITIAL  (' '),
                           3  IDBMSCOM_AREA,
                             5  IDBMSCOM (100) CHARACTER (1),
                           3  DIRECT_DBKEY FIXED BINARY (31),
                           3  DCBMSCOM_AREA,
                             5  DCBMSCOM (100) CHARACTER (1),
                           3  DCCALIGN_AREA,
                             5  FILLER0001 CHARACTER (4),
                             5  DCCALIGN FLOAT BINARY (53),
                             5  FILLER0002 CHARACTER (8);
      13   1  0  DECLARE 1  SSC_ERRSAVE_AREA BASED(ADDR(SUBSCHEMA_CTRL.DCCALIGN_AREA)),
                           3  SSC_ERRSTAT_SAVE CHARACTER (4),
                           3  SSC_DMLSEQ_SAVE FIXED BINARY (31),
                           3  DML_SEQUENCE FIXED BINARY (31),
                           3  RECORD_OCCUR FIXED BINARY (31),
                           3  SUBSCHEMA_CTRL_END CHARACTER (4);
      14   1  0  DECLARE 1  DCCFN_AREA BASED(ADDR(SUBSCHEMA_CTRL.DCBMSCOM_AREA)),
                           3  FILLER0003 CHARACTER (44),
                           3  DCCSTR1 CHARACTER (16),
                           3  DCCNUM1 FIXED BINARY (31),
                           3  DCCNUM2 FIXED BINARY (31),
                           3  DCCNUM3 FIXED BINARY (31),
                           3  DCCFLG1 FIXED BINARY (15),
                           3  DCCFLG2 FIXED BINARY (15),
                           3  DCCFLG3 FIXED BINARY (15),
                           3  DCCFLG4 FIXED BINARY (15),
                           3  DCCFLG5 FIXED BINARY (15),
                           3  DCCFLG6 FIXED BINARY (15),
                           3  FILLER0004 CHARACTER (4),
                           3  DCCDBLWK CHARACTER (8);
      15   1  0  DECLARE 1  DCCPT_AREA BASED(ADDR(SUBSCHEMA_CTRL.DCBMSCOM_AREA)),
                           3  FILLER0005 CHARACTER (60),
                           3  DCCPT1 POINTER,
                           3  DCCPT2 POINTER;
      16   1  0  DECLARE 1  DCCPN_AREA BASED(ADDR(SUBSCHEMA_CTRL.DCBMSCOM_AREA)),
                           3  FILLER0006 CHARACTER (44),
                           3  DCCPNUM1 FIXED DECIMAL(11,0),
                           3  FILLER0007 CHARACTER (10),
                           3  DCCPNUM2 FIXED DECIMAL(7,0);
      17   1  0  DECLARE 1  DCCSTR_AREA3 BASED(ADDR(SUBSCHEMA_CTRL.DCBMSCOM_AREA)),
                           3  FILLER0008 CHARACTER (44),
                           3  DCCSTR4 CHARACTER (4),
                           3  DCCSTR5 CHARACTER (4),
                           3  DCCSTR3 CHARACTER (8);
      18   1  0  DECLARE 1  DCCSTR_AREA2 BASED(ADDR(SUBSCHEMA_CTRL.DCBMSCOM_AREA)),
                           3  FILLER0009 CHARACTER (44),
                           3  DCCSTR2 CHARACTER (8);
      19   1  0  DECLARE 1  DCCSTR_AREA1 BASED(ADDR(SUBSCHEMA_CTRL.DCBMSCOM_AREA)),
                           3  FILLER0010 CHARACTER (44),
                           3  DCCSTR6 CHARACTER (32),
                           3  DCCNUH1 FIXED BINARY (15),
                           3  FILLER0011 CHARACTER (2),
                           3  DC_ABEND_CODE CHARACTER (4);
      20   1  0  DECLARE 1  DCCPLI_DEFS BASED(ADDR(SUBSCHEMA_CTRL.DCBMSCOM_AREA)),
```

```
                     3  DCCR14SV FIXED BINARY (31),
                     3  DCCPARMS (10) FIXED BINARY (31);
21   1  0  DECLARE 1  AREA_RNAME BASED(ADDR(SUBSCHEMA_CTRL.AREA_NAME)),
                     3  SSC_DNO CHARACTER (8),
                     3  SSC_DNA CHARACTER (8);
22   1  0  DECLARE 1  RRECORD_NAME BASED(ADDR(SUBSCHEMA_CTRL.RECORD_NAME)),
                     3  SSC_NODN CHARACTER (8),
                     3  SSC_DBN CHARACTER (8);

                                                                              /*
            INCLUDE IDMS (EMPLOYEE);
                                                                              */
23   1  0  DECLARE 1  EMPLOYEE,
                    2  EMP_ID_0415 PICTURE '(4)9',
                    2  EMP_NAME_0415,
                     3  EMP_FIRST_NAME_0415 CHARACTER (10),
                     3  EMP_LAST_NAME_0415 CHARACTER (15),
                    2  EMP_ADDRESS_0415,
                     3  EMP_STREET_0415 CHARACTER (20),
                     3  EMP_CITY_0415 CHARACTER (15),
                     3  EMP_STATE_0415 CHARACTER (2),
                     3  EMP_ZIP_0415,
                      4  EMP_ZIP_FIRST_FIVE_0415 CHARACTER (5),
                      4  EMP_ZIP_LAST_FOUR_0415 CHARACTER (4),
                    2  EMP_PHONE_0415 PICTURE '(10)9',
                    2  STATUS_0415 CHARACTER (2),
                    2  SS_NUMBER_0415 PICTURE '(9)9',
                    2  START_DATE_0415,
                     3  START_YEAR_0415 PICTURE '(2)9',
                     3  START_MONTH_0415 PICTURE '(2)9',
                     3  START_DAY_0415 PICTURE '(2)9',
                    2  TERMINATION_DATE_0415,
                     3  TERMINATION_YEAR_0415 PICTURE '(2)9',
                     3  TERMINATION_MONTH_0415 PICTURE '(2)9',
                     3  TERMINATION_DAY_0415 PICTURE '(2)9',
                    2  BIRTH_DATE_0415,
                     3  BIRTH_YEAR_0415 PICTURE '(2)9',
                     3  BIRTH_MONTH_0415 PICTURE '(2)9',
                     3  BIRTH_DAY_0415 PICTURE '(2)9',
                    2  FILLER0012 CHARACTER (2),
                    2  FILLER0013 CHARACTER (4);
                                                                              /*
            INCLUDE IDMS (MAP_CONTROLS);
                                                                              */
24   1  0  DECLARE 1  MRB_EMPLMAP,
                       5  MRB_EMPLMAP_ID CHARACTER (8),
                       5  MRB_EMPLMAP_MCOMP_VER,
                          8  MRB_EMPLMAP_MCOMP_DATE CHARACTER (8),
                          8  MRB_EMPLMAP_MCOMP_TIME CHARACTER (6),
                          8  MRB_EMPLMAP_MCOMP_VERID CHARACTER (2),
                       5  MRB_EMPLMAP_SUBSCHEMA CHARACTER (8),
                       5  MRB_EMPLMAP_FLGS (4) CHARACTER (1),
                       5  FILLER0014 CHARACTER (6),
                       5  MRB_EMPLMAP_NFLDS FIXED BINARY (15),
                       5  MRB_EMPLMAP_NRECS FIXED BINARY (15),
                       5  MRB_EMPLMAP_RECOF FIXED BINARY (15),
                       5  MRB_EMPLMAP_PERM_CURSOR CHARACTER (2),
                       5  MRB_EMPLMAP_TEMP_CURSOR CHARACTER (2),
                       5  MRB_EMPLMAP_PERM_WCC CHARACTER (1),
                       5  MRB_EMPLMAP_TEMP_WCC CHARACTER (1),
                       5  MRB_EMPLMAP_CURSOR CHARACTER (2),
                       5  MRB_EMPLMAP_AID CHARACTER (1),
                       5  MRB_EMPLMAP_INPUT_FLGS CHARACTER (1),
                       5  MRB_EMPLMAP_SEGVIEW CHARACTER (1),
                       5  FILLER0015 CHARACTER (1),
                       5  MRB_EMPLMAP_MREO FIXED BINARY (15),
                       5  MRB_EMPLMAP_ERR_CNT FIXED BINARY (15),
                       5  MRB_EMPLMAP_ATTR_FLGS (4) CHARACTER (1),
                       5  MRB_EMPLMAP_CURR_MFLD FIXED BINARY (15),
                       5  MRB_EMPLMAP_XTYP CHARACTER (1),
                       5  FILLER0016 CHARACTER (1),
                       5  MRB_EMPLMAP_MRE_XLEN FIXED BINARY (15),
                       5  MRB_EMPLMAP_MRB_XLEN FIXED BINARY (15),
                       5  MRB_EMPLMAP_MRE (8),
                          8  MRB_EMPLMAP_MRE_FLGS (8) CHARACTER (1),
                          8  MRB_EMPLMAP_MRE_INLEN FIXED BINARY (15),
                          8  MRB_EMPLMAP_MRE_PAD_CHAR (2) CHARACTER (1),
                          8  MRB_EMPLMAP_MRE_FLG2 (2) CHARACTER (1),
                       5  MRB_EMPLMAP_RECS (1) FIXED BINARY (31),
                       5  MRB_EMPLMAP_END CHARACTER (1),
                       5  MRB_EMPLMAP_MRE_SUB FIXED BINARY (15);

            /*        PROCESSING FOLLOWS                          */

25   1  0   MAIN_LINE: BEGIN;
                                     /* ESTABLISH ADDRESSABILITY FOR */
                                                                              /*
                       BIND MAP (EMPLMAP);                                      DMLP0001
                                                                              */
26   2  0                      /* IDMS PL/I DML EXPANSION */           DO;
27   2  1                      DML_SEQUENCE=1;
28   2  1                      DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
29   2  1                      CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (90)
                                                  ,MRB_EMPLMAP
                                                  ,MRB_EMPLMAP_END
30   2  1                                                        ); END;
31   2  0                      STRING(MRB_EMPLMAP_MCOMP_VER)=
                                   '11/04/87172444R2';
32   2  0                      MRB_EMPLMAP_SUBSCHEMA=
                                   'EMPSS01';
33   2  0                      MRB_EMPLMAP_ID=
                                   'EMPLMAP';
34   2  0                      MRB_EMPLMAP_NFLDS=
                                   8;
35   2  0                      MRB_EMPLMAP_NRECS=
```

```
                                  1;
 36   2  0                    MRB_EMPLMAP_RECOF=
                                  112;
 37   2  0                    MRB_EMPLMAP_MREO=
                                  76;
 38   2  0                    MRB_EMPLMAP_XTYP=
                                  '0';
 39   2  0                    MRB_EMPLMAP_MRE_XLEN=
                                  0;
 40   2  0                    MRB_EMPLMAP_MRB_XLEN=
                                  0;
 41   2  0                    MRB_EMPLMAP_SEGVIEW=
                                  'N';
 42   2  0           CALL IDMS_STATUS;
                                                                     /*
                     BIND MAP (EMPLMAP) RECORD (EMPLOYEE);             DMLP0002
                                                                     */
 43   2  0                    /* IDMS PL/I DML EXPANSION */        DO;
 44   2  1                    DML_SEQUENCE=2;
 45   2  1                    DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
 46   2  1                    CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (91)
                                                  ,MRB_EMPLMAP_RECS (1)
                                                  ,EMPLOYEE
 47   2  1                                                      ); END;
 48   2  0           CALL IDMS_STATUS;
                                        /* DETERMINE THE TASK CODE */
                                                                     /*
                     ACCEPT TASK CODE INTO (TASK_CODE);                DMLP0003
                                                                     */
 49   2  0                    /* IDMS PL/I DML EXPANSION */        DO;
 50   2  1                    DML_SEQUENCE=3;
 51   2  1                    DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
 52   2  1                    DCCNUM1=1;
 53   2  1                    CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (2)
 54   2  1                                                      ); END;
 55   2  0                    TASK_CODE=DCCSTR6;
 56   2  0           CALL IDMS_STATUS;

 57   2  0           IF TASK_CODE = EMPDISP
                     THEN CALL FIRST_TIME;
 58   2  0           IF TASK_CODE = EMPDISP2
                     THEN CALL SECOND_TIME;



                                 /* OTHERWISE RETURN TO IDMS-DC */
                                                                     /*
                     DC RETURN;                                        DMLP0004
                                                                     */
 59   2  0                    /* IDMS PL/I DML EXPANSION */        DO;
 60   2  1                    DML_SEQUENCE=4;
 61   2  1                    DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
 62   2  1                    CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (19)
 63   2  1                                                      ); END;

 64   2  0  FIRST_TIME: PROC;
                                                                     /*
               MODIFY MAP (EMPLMAP)                                    DMLP0005
                  FOR ALL BUT DFLD (EMP_ID_0415)
                  ATTRIBUTES PROTECTED;
                                                                     */
 65   3  0                    /* IDMS PL/I DML EXPANSION */        DO;
 66   3  1                    DML_SEQUENCE=5;
 67   3  1                    DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
 68   3  1                    DCCNUM1=8;
 69   3  1                    DCCFLG1=768;
 70   3  1                    DCCFLG3=0;
 71   3  1                    DCCFLG4=0;
 72   3  1                    CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (93)
                                                  ,MRB_EMPLMAP
                                                  ,MRB_EMPLMAP_MRE (1)
 73   3  1                                                      ); END;

                                                                     /*
               MAP OUT(EMPLMAP)                                        DMLP0006
                  IO OUTPUT DATA YES NEWPAGE;
                                                                     */
 74   3  0                    /* IDMS PL/I DML EXPANSION */        DO;
 75   3  1                    DML_SEQUENCE=6;
 76   3  1                    DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
 77   3  1                    DCCFLG1=5;
 78   3  1                    DCCFLG2=16;
 79   3  1                    DCCFLG3=1;
 80   3  1                    DCCFLG4=0;
 81   3  1                    DCCFLG5=0;
 82   3  1                    DCCFLG6=1;
 83   3  1                    CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (34)
                                                  ,MRB_EMPLMAP
 84   3  1                                                      ); END;
 85   3  0     CALL IDMS_STATUS;
                                                                     /*
               DC RETURN NEXT TASK CODE(EMPDISP2);                     DMLP0007
                                                                     */
 86   3  0                    /* IDMS PL/I DML EXPANSION */        DO;
 87   3  1                    DML_SEQUENCE=7;
 88   3  1                    DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
 89   3  1                    DCCSTR2=EMPDISP2;
 90   3  1                    DCCFLG1=128;
 91   3  1                    CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (19)
 92   3  1                                                      ); END;
```

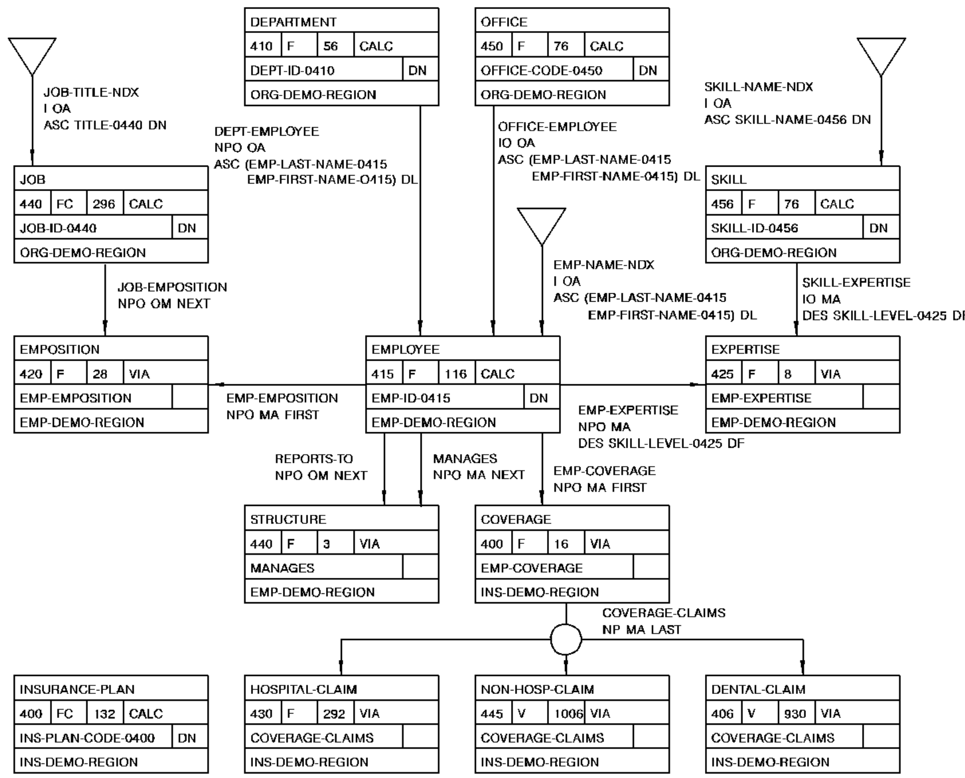
```
 93   3  0   END FIRST_TIME;

 94   2  0   SECOND_TIME: PROC;
                                                                            /*
                MAP IN (EMPLMAP)                                              DMLP0008
                   IO INPUT DATA YES;
                                                                            */
 95   3  0                   /* IDMS PL/I DML EXPANSION */          DO;
 96   3  1                   DML_SEQUENCE=8;
 97   3  1                   DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
 98   3  1                   DCCFLG1=6;
 99   3  1                   DCCFLG2=4;
100   3  1                   DCCFLG3=0;
101   3  1                   DCCFLG4=0;
102   3  1                   DCCFLG5=0;
103   3  1                   DCCFLG6=0;
104   3  1                   CALL IDMSPLI (SUBSCHEMA_CTRL
                                               ,DCBMSCOM (34)
                                               ,MRB_EMPLMAP
105   3  1                                                       ); END;
106   3  0      CALL IDMS_STATUS;
                                           /* CHECK WHICH PF KEY WAS PRESSED */
                                                                            /*
                INQUIRE MAP(EMPLMAP)                                          DMLP0009
                    MOVE AID TO (DC_AID_IND_V);
                                                                            */
107   3  0                   /* IDMS PL/I DML EXPANSION */          DO;
108   3  1                   DML_SEQUENCE=9;
109   3  1                   DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
110   3  1                   DCCNUM1=7;
111   3  1                   CALL IDMSPLI (SUBSCHEMA_CTRL
                                               ,DCBMSCOM (92)
                                               ,MRB_EMPLMAP
112   3  1                                                       ); END;
113   3  0                   DC_AID_IND_V=DCCSTR2;

                                           /* STOP IF PA1 (%) WAS PRESSED */
114   3  0      IF DC_AID_IND_V = '%'
                    THEN                                                      DMLP0010
                                                                            /*
                         DC RETURN;
                                                                            */
                             /* IDMS PL/I DML EXPANSION */          DO;
115   3  1                   DML_SEQUENCE=10;
116   3  1                   DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
117   3  1                   CALL IDMSPLI (SUBSCHEMA_CTRL
                                               ,DCBMSCOM (19)
118   3  1                                                       ); END;

                                                                            /*
                BIND RUN_UNIT;                                                DMLP0011
                                                                            */
119   3  0                   /* IDMS PL/I DML EXPANSION */          DO;
120   3  1                   DML_SEQUENCE=11;
121   3  1                   DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
122   3  1                   CALL IDMSPLI (SUBSCHEMA_CTRL
                                               ,IDBMSCOM (59)
                                               ,SUBSCHEMA_CTRL
                                               ,'EMPSS01 '
123   3  1                                                       ); END;
124   3  0      CALL IDMS_STATUS;
                                                                            /*
                BIND RECORD (EMPLOYEE);                                       DMLP0012
                                                                            */
125   3  0                   /* IDMS PL/I DML EXPANSION */          DO;
126   3  1                   DML_SEQUENCE=12;
127   3  1                   DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
128   3  1                   CALL IDMSPLI (SUBSCHEMA_CTRL
                                               ,IDBMSCOM (48)
                                               ,'EMPLOYEE        '
                                               ,EMPLOYEE
129   3  1                                                       ); END;
130   3  0      CALL IDMS_STATUS;
                                                                            /*
                READY AREA (EMP_DEMO_REGION);                                 DMLP0013
                                                                            */
131   3  0                   /* IDMS PL/I DML EXPANSION */          DO;
132   3  1                   DML_SEQUENCE=13;
133   3  1                   DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
134   3  1                   CALL IDMSPLI (SUBSCHEMA_CTRL
                                               ,IDBMSCOM (37)
                                               ,'EMP-DEMO-REGION '
135   3  1                                                       ); END;
136   3  0      CALL IDMS_STATUS;
                                         /* OBTAIN THE RECORD */
                                                                            /*
                OBTAIN CALC RECORD (EMPLOYEE);                                DMLP0014
                                                                            */
137   3  0                   /* IDMS PL/I DML EXPANSION */          DO;
138   3  1                   DML_SEQUENCE=14;
139   3  1                   DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
140   3  1                   CALL IDMSPLI (SUBSCHEMA_CTRL
                                               ,IDBMSCOM (32)
                                               ,'EMPLOYEE        '
                                               ,IDBMSCOM (43)
141   3  1                                                       ); END;
142   3  0      IF ERROR_STATUS = '0326' THEN CALL NO_EMP;
143   3  0      CALL IDMS_STATUS;
                                                                            /*
                FINISH;                                                       DMLP0015
                                                                            */
144   3  0                   /* IDMS PL/I DML EXPANSION */          DO;
145   3  1                   DML_SEQUENCE=15;
146   3  1                   DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
147   3  1                   CALL IDMSPLI (SUBSCHEMA_CTRL
                                               ,IDBMSCOM (2)
```

```
148   3  1                                                         ); END;
149   3  0      CALL IDMS_STATUS;
                                    /* TRANSMIT THE DATA BACK TO THE SCREEN */
                                                                                 /*
                MAP OUT(EMPLMAP)                                                   DMLP0016
                   IO OUTPUT DATA YES NEWPAGE
                   MESSAGE(DISPLAY_MSG) LENGTH(36);
                                                                                 */
150   3  0                       /* IDMS PL/I DML EXPANSION */           DO;
151   3  1                       DML_SEQUENCE=16;
152   3  1                       DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
153   3  1                       DCCFLG1=5;
154   3  1                       DCCFLG2=16;
155   3  1                       DCCFLG3=1;
156   3  1                       DCCFLG4=4;
157   3  1                       DCCFLG5=0;
158   3  1                       DCCFLG6=1;
159   3  1                       CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (34)
                                                  ,MRB_EMPLMAP
                                                  ,DISPLAY_MSG
                                                  ,DCBMSCOM (36)
160   3  1                                                         ); END;
161   3  0      CALL IDMS_STATUS;
                                                                                 /*
                DC RETURN NEXT TASK CODE(EMPDISP2);                                DMLP0017
                                                                                 */
162   3  0                       /* IDMS PL/I DML EXPANSION */           DO;
163   3  1                       DML_SEQUENCE=17;
164   3  1                       DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
165   3  1                       DCCSTR2=EMPDISP2;
166   3  1                       DCCFLG1=128;
167   3  1                       CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (19)
168   3  1                                                         ); END;

169   3  0   END SECOND_TIME;

170   2  0   NO_EMP: PROC;
                                    /* DO THIS IF EMPLOYEE NOT FOUND */
                                                                                 /*
               MAP OUT(EMPLMAP)                                                    DMLP0018
                  IO OUTPUT DATA YES NEWPAGE
                  MESSAGE(NOT_FOUND_MSG) LENGTH(37);
                                                                                 */
171   3  0                       /* IDMS PL/I DML EXPANSION */           DO;
172   3  1                       DML_SEQUENCE=18;
173   3  1                       DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
174   3  1                       DCCFLG1=5;
175   3  1                       DCCFLG2=16;
176   3  1                       DCCFLG3=1;
177   3  1                       DCCFLG4=4;
178   3  1                       DCCFLG5=0;
179   3  1                       DCCFLG6=1;
180   3  1                       CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (34)
                                                  ,MRB_EMPLMAP
                                                  ,NOT_FOUND_MSG
                                                  ,DCBMSCOM (37)
181   3  1                                                         ); END;
182   3  0     CALL IDMS_STATUS;
                                                                                 /*
               DC RETURN NEXT TASK CODE(EMPDISP2);                                 DMLP0019
                                                                                 */
183   3  0                       /* IDMS PL/I DML EXPANSION */           DO;
184   3  1                       DML_SEQUENCE=19;
185   3  1                       DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
186   3  1                       DCCSTR2=EMPDISP2;
187   3  1                       DCCFLG1=128;
188   3  1                       CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (19)
189   3  1                                                         ); END;
190   3  0   END NO_EMP;

                                                                                 /*
                        INCLUDE IDMS (IDMS_STATUS);
                                                                                 */
191   2  0  IDMS_STATUS:  PROC;
            /* THE IDMS_STATUS PROCEDURE MAY BE CALLED BY THE USER AFTER */
            /* EACH IDMS COMMAND HAS BEEN ISSUED AND CHECKS HAVE BEEN    */
            /* MADE FOR ANY EXPECTED NON_ZERO ERROR STATUS CONDITIONS.   */
            /* IT DETECTS A NON_ZERO ERROR_STATUS AND TERMINATES THE     */
            /* PROGRAM WITH A SNAP OF THE SUBSCHEMA_CTRL AREA AND AN     */
            /* ABEND WITH THE ERROR_STATUS AS THE ABEND CODE.            */
192   3  0      IF ERROR_STATUS='0000' THEN GOTO END_STATUS;
193   3  0      SSC_ERRSTAT_SAVE=ERROR_STATUS; /* SAVE THE ERROR_STATUS */
194   3  0      SSC_DMLSEQ_SAVE=DML_SEQUENCE; /* SAVE DML_SEQUENCE     */
            /* SNAP THE SUBSCHEMA_CTRL AREA */
                                                                                 /*
                SNAP FROM (SUBSCHEMA_CTRL) TO (SUBSCHEMA_CTRL_END);
                                                                                 */
195   3  0                       /* IDMS PL/I DML EXPANSION */           DO;
196   3  1                       DML_SEQUENCE=20;
197   3  1                       DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
198   3  1                       CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (22)
                                                  ,DCCSTR1
                                                  ,DCCSTR1
                                                  ,DCCSTR1
                                                  ,SUBSCHEMA_CTRL
                                                  ,SUBSCHEMA_CTRL_END
                                                  ,DCBMSCOM (1)
199   3  1                                                         ); END;
            /* ABEND */
                                                                                 /*
                ABEND CODE (SSC_ERRSTAT_SAVE);
```

```
                                                                                 */
 200   3  0                       /* IDMS PL/I DML EXPANSION */           DO;
 201   3  1                       DML_SEQUENCE=21;
 202   3  1                       DCCFLG1,DCCFLG2,DCCNUM1,DCCNUM2=0;
 203   3  1                       DCCSTR4=SSC_ERRSTAT_SAVE;
 204   3  1                       DCCFLG1=2;
 205   3  1                       CALL IDMSPLI (SUBSCHEMA_CTRL
                                                  ,DCBMSCOM (1)
 206   3  1                                                             ); END;
 207   3  0  END_STATUS: END;
 208   2  0   END MAIN_LINE; /* END MAIN_LINE */
 209   1  0   END EMPDISP;
```

E.5 EMPLOYEE database definition

The following is a data structure diagram for the EMPLOYEE database. Most of the examples used in this manual (including the sample programs in this appendix) use the EMPLOYEE database.



Appendix F. Considerations for IBM Language Environment

F.1 Overview

What is IBM Language Environment (LE)?: LE is a runtime environment that replaces the language-specific runtime environments that existed previously. For example, PL/I had its own runtime environment; COBOL II had another. CA-IDMS can execute programs that are designed to use the LE runtime environment. It can also execute programs compiled with pre-LE compilers that use the LE runtime environment.

Note: This appendix only applies to runtime support in CA-IDMS/DC. It does not apply to batch or CICS programs that access CA-IDMS.

Language Environment has had several names for different operating systems and release levels. The term "LE" will be used in this document to refer to the any of the following unless otherwise noted:

- LE/370
- LE for OS/390 and VM/ESA
- LE for VSE/ESA

How can you use LE with CA-IDMS/DC?: To execute online programs using the LE run-time libraries, follow these steps to bring up your CA-IDMS environment:

1. Ensure that the CA-IDMS system has been generated with a 24-bit reentrant pool that is large enough to contain the IBM-supplied LE application program interface module CEEPIPI. The size of this module is approximately 100K.
2. Ensure that the CA-IDMS system has been generated with an XA reentrant pool that is large enough to maintain residence for several IBM-supplied LE support modules. Allow 1 megabyte for these programs.
3. Include the LE runtime load libraries in the CDMSLIB loadlib concatenation before any other IBM language loadlibs that you are using.

F.2 Considerations about LE runtime

Running pre-LE programs: There are restrictions that apply when you run pre-LE programs under LE runtime within CA-IDMS/DC. Pre-LE programs are programs that were compiled with a non-LE compliant compiler, such as PL/I Release 2.3.

Some of these restrictions are already documented elsewhere in the DML Reference manuals. Additional restrictions for LE are:

- Programs compiled under PL/I Release 2.3 and earlier must run without storage protection.
- The IBM LE support module CEEPIPI must be loaded once before any PL/I program is run. This is most easily done by defining CEEPIPI as RESIDENT in the CA-IDMS/DC sysgen using the following syntax:

```
ADD PROGRAM CEEPIPI CONCURRENT ENABLED LANGUAGE ASSEMBLER
NONOVERLABYABLE PROGRAM PROTECT REENTRANT RESIDENT REUSABLE .
```

- Restrictions mentioned in the IBM documentation apply.

Note: Running pre-LE programs with LE runtime can degrade performance in some circumstances. If you notice poor performance you should consider recompiling the programs with the newer compiler.

Running LE programs: LE programs are programs that were compiled with a LE-compliant compiler. CA-IDMS/DC supports these LE-compliant compilers:

- PL/I for VM/ESA
- PL/I for OS/390

For convenience, PL/I programs compiled with an LE-compliant compiler are referred to as "LE PL/I" programs below.

F.3 Running LE-compliant compiler programs under CA-IDMS/DC

This section describes what you need to do to compile, link, and run a program compiled with an LE-compliant compiler.

General preparation: The next paragraph describes how to prepare LE-compiled programs for use with CA-IDMS/DC:

For non-reentrant PL/I programs compiled under Release 2.3 or earlier, you must specify OPTIONS (MAIN) in the PL/I PROCEDURE statement for the entry procedure. For reentrant PL/I Release 2.3 or earlier programs, you must specify OPTIONS (MAIN,REENTRANT). For AD/CYCLE (LE-COMPLIANT), PL/I programs, you must specify OPTIONS (REENTRANT,FETCHABLE).

Note: RHDCLENT/RHDCLINT, required in earlier releases, is not needed for CA-IDMS/DC/UCF at release levels 14.1 and above.

Run-time options: The IBM Language Environment provides numerous options which control how programs operate at run-time. The default values are designed to be suitable in a batch environment. Therefore, it is necessary to modify some values for applications which are to run in a CA-IDMS/DC/UCF online system.

Note: As stated in the introduction, the information in this appendix does not apply to programs which run in a CICS or other region even if they access CA-IDMS via DML or SQL commands. It does apply to programs which run a CA-IDMS/DC/UCF online system which are invoked from another front-end via UCF, such as a CA-ADS application which is accessed via UCFCICS from a CICS front-end.

The IBM Language Environment provides a number of ways to specify run-time options. The following two methods are supported for CA-IDMS/DC PL/I online programs:

- Modify, assemble, and link the IBM-supplied CEEUOPT module. Link the resulting module with each application program.

  The following source is provided as a sample for assembling a CEEUOPT module for use with table procedures or online CA-IDMS/DC/UCF programs compiled in an IBM Language Environment for OS/390 and VM/ESA. The sample options may not be appropriate at all sites, but they provide guidelines which will be valid for most table procedures and online programs. These guidelines are for use with CA-IDMS Release 10.21 Premium Support and above.

  Note that various storage size run-time options should be reduced from the normal IBM installed defaults. Storage allocation in a CA-IDMS/DC/UCF system is most efficient (in terms of CPU utilization) if LE storage parameters are specified as sixteen bytes less than a multiple of 4096. Smaller sizes should be used for some parameters to avoid wasting storage.

See the *IBM Language Environment for OS/390 Customization* manual for information on creating an application-specific run-time options module. Once a CEEUOPT load module is created, it should be linked with each table procedure or online CA-IDMS/DC/UCF program compiled under Language Environment for OS/390 and VM/ESA.

```
*/*******************************************************************/
*/* Parameters which are not shown below do not need to be         */
*/* coded.  The IBM default values should be appropriate for        */
*/* those parameters.                                               */
*/*                                                                 */
CEEUOPT  CSECT                                                        CEE00160
CEEUOPT  AMODE ANY                                                    CEE00170
CEEUOPT  RMODE ANY                                                    CEE00180
         CEEXOPT ABPERC=(NONE),                                      XCEE00190
               ABTERMENC=(RETCODE),                                  XCEE00200
               AIXBLD=(OFF),                                         XCEE00210
               ALL31=(ON),                                           XCEE00220
               ANYHEAP=(2032,4080,ANYWHERE,FREE),                    XCEE00230
               BELOWHEAP=(496,496,FREE),                             XCEE00240
               CBLOPTS=(ON),                                         XCEE00250
               CBLPSHPOP=(OFF),                                      XCEE00260
               CBLQDA=(OFF),                                         XCEE00270
               CHECK=(OFF),                                          XCEE00280
               COUNTRY=(US),                                         XCEE00290
               DEBUG=(OFF),                                08/06/97| XCEE00300
               DEPTHCONDLMT=(10),                                    XCEE00310
               ENVAR=(''),                                           XCEE00320
               ERRCOUNT=(20),                                        XCEE00330
               ERRUNIT=(6),                                          XCEE00340
               FILEHIST=(ON),                                        XCEE00350
               HEAP=(2032,4080,ANYWHERE,KEEP,2032,2032),             XCEE00360
               INQPCOPN=(ON),                                        XCEE00370
               INTERRUPT=(OFF),                                      XCEE00380
               LIBSTACK=(496,496,FREE),                              XCEE00390
               MSGFILE=(SYSOUT,FBA,121,0),                           XCEE00400
               MSGQ=(15),                                            XCEE00410
               NATLANG=(ENU),                                        XCEE00420
               NOAUTOTASK=,                                          XCEE00430
               NONONIPTSTACK=(100,2032,BELOW,KEEP),                  XCEE00440
               NOTEST=(ALL,*,PROMPT,INSPPREF),                       XCEE00450
               NOUSRHDLR=(),                                         XCEE00460
               OCSTATUS=(ON),                                        XCEE00470
               PC=(OFF),                                             XCEE00480
               PLITASKCOUNT=(20),                                    XCEE00490
               POSIX=(OFF),                                          XCEE00500
               PRTUNIT=(6),                                          XCEE00510
               PUNUNIT=(7),                                          XCEE00520
               RDRUNIT=(5),                                          XCEE00530
               RECPAD=(OFF),                                         XCEE00540
               RPTOPTS=(OFF),                                        XCEE00550
               RPTSTG=(OFF),                                         XCEE00560
               RTEREUS=(OFF),                                        XCEE00570
               SIMVRD=(OFF),                                         XCEE00580
               STACK=(2032,8176,ANY,KEEP),                           XCEE00590
               STORAGE=(NONE,NONE,NONE,4080),                        XCEE00600
               TERMTHDACT=(QUIET),                                   XCEE00610
               THREADHEAP=(0100,0100,,ANYWHERE,KEEP),                XCEE00620
               TRACE=(OFF,0000,DUMP,LE=0),                           XCEE00630
               TRAP=(ON),                                            XCEE00640
               UPSI=(00000000),                                      XCEE00650
               VCTRSAVE=(OFF),                                       XCEE00660
               XUFLOW=(AUTO)                                          CEE00670
         DC    C'5688-198 (C) COPYRIGHT IBM CORP. 1991, 1995. '       CEE00680
         DC    C'LICENSED MATERIALS - PROPERTY OF IBM'                CEE00690
         END                                                          CEE00700
```

Note: Refer to the *IBM Language Environment for OS/390 Customization* manual for more information.

- Assemble and link a specialized CEEDOPT module. If this method is chosen, special copies of the IBM modules CEEBINIT and CEEPIPI must be maintained for use with online CA-IDMS/DC/UCF systems only. This raises special maintenance considerations:

  – Removes the necessity of linking a CEEUOPT with each program.

  – Allows changes to run-time options for an entire CA-IDMS/DC system or systems without the need to relink individual application programs.

  When running PL/I programs on an online CA-IDMS/DC/UCF system with IBM's Language Environment, excessive amounts of storage will be used. This is the result of the default run-time options set up during the installation of Language Environment.

  In addition, some default LE run-time options may cause unnecessary CPU consumption regardless of the language level. A few default options are not compatible with some or all releases of CA-IDMS/DC/UCF.

  The following information documents how to tailor the run-time options for a specific CA-IDMS/DC/UCF system. The options shown below have been determined to be appropriate for most sites. Individual sites may wish to modify certain options because applications are known to use a particular amount of storage or for some other purpose.

  The run-time options can be tailored by compiling a CEEXOPT macro and creating a USERMOD based on the sample in member CEEWDOPT in the SCEESAMP library. Refer to the *IBM Language Environment for OS/390 Customization* manual for more details.

  Note: This procedure is not recommended for most sites.

  If the CEEWDOPT sample JCL is used, the run-time options will be changed for all applications which use LE run-time support, not just online CA-IDMS/DC/UCF applications. The options which are appropriate for online applications are likely to be inappropriate for CA-IDMS batch applications or non-IDMS applications.

  An alternative procedure which allows for tailoring run-time options for only CA-IDMS/DC/UCF online applications is presented below. If this procedure is followed, it will also remove the necessity for linking a tailored CEEUOPT module with each online CA-IDMS/DC/UCF application program compiled under LE compilers.

  Notes:

  – The parameters shown below are valid for LE Version 1.8.0. Other releases of LE may have more or fewer parameters.

    All parameters supplied in the site IBM SCEESAMP library member CEEDOPT should be specified. Use the IBM-supplied defaults for any parameters which are not shown below. Use the defaults in this appendix for any parameters which are shown below and which also appear in the SCEESAMP member. Remove any parameters or subparameters which are shown below but which do not exist in the SCEESAMP member for your site.

Be certain that the macro library used to compile CEEDOPT is from the same release as the CEEPIPI and CEEPLPKA load modules which are to be modified.

– The parameters shown below all specify the "OVR" option. This allows individual applications to use different options by linking those applications with a tailored version of CEEUOPT.

  Alternatively, a site may opt to use the "NOOVR" option for some or all of the parameters below. Then the CEEDOPT options will take effect regardless of the options specified in any CEEUOPT module linked with an individual application. This may be useful if a site wishes to change the run-time options for all online applications in one or more CA-IDMS/DC/UCF systems without relinking each application.

At this time, IBM does not have SMP support for changing the run-time options for some systems and not for others. However, the options can be modified for a particular DC system if two IBM modules are relinked outside of SMP using a tailored CEEDOPT CSECT. IBM technical support has confirmed that this technique is valid.

The two modules are CEEBINIT and CEEPIPI. They must be linked into a load library which is included in the CDMSLIB concattenation in the CA-IDMS/DC/UCF startup JCL AHEAD of the IBM SCEERUN load library. This library should be included ONLY in the startup JCL for online systems.

The sample JCL below can be used as a basis for making these changes. Note that after any IBM LE maintenance is performed, the JCL below should be rerun to include the new IBM maintenance.

The sample macro shown below is appropriate for most IDMS online applications and can be used to build the appropriate CEEDOPT CSECT whether the changes are made inside or outside of SMP.

```
//*********************************************************************
//************************  ASSEMBLE CEEDOPT  *************************
//*********************************************************************
//ASM       EXEC PGM=ASMA90,PARM='DECK,NOOBJECT'
//SYSLIB    DD DISP=SHR,DSN=SYS1.MACLIB
//          DD DISP=SHR,DSN=SYS1.AMODGEN
//          DD DISP=SHR,DSN=CEE.SCEEMAC   **IBM CEE MACLIB
//SYSPRINT DD SYSOUT=*
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(2,1)),DSN=&&WORK1
//SYSUT2    DD UNIT=SYSDA,SPACE=(CYL,(2,1)),DSN=&&WORK2
//SYSUT3    DD UNIT=SYSDA,SPACE=(CYL,(2,1)),DSN=&&WORK3
//SYSPUNCH DD DISP=(NEW,PASS),SPACE=(TRK,(10,5)),UNIT=SYSDA,
//          DSN=&&OBJECT
//SYSIN     DD *
*/******************************************************************/ 00100000
CEEDOPT  CSECT                                                        00110000
CEEDOPT  AMODE ANY                                                    00120000
CEEDOPT  RMODE ANY                                                    00130000
         CEEXOPT ABPERC=((NONE),OVR),                                X00140000
               ABTERMENC=((RETCODE),OVR),                            X00150000
               AIXBLD=((OFF),OVR),                                   X00160000
               ALL31=((ON),OVR),                                     X00170000
               ANYHEAP=((2032,4080,ANYWHERE,FREE),OVR),              X00180000
               BELOWHEAP=((496,496,FREE),OVR),                       X00190000
               CBLOPTS=((ON),OVR),                                   X00200000
               CBLPSHPOP=((OFF),OVR),                                X00210000
               CBLQDA=((OFF),OVR),                                   X00220000
               CHECK=((OFF),OVR),                                    X00230000
               COUNTRY=((US),OVR),                                   X00240000

               DEBUG=((OFF),OVR),                                    X00250000
               DEPTHCONDLMT=((10),OVR),                              X00260000
               ENVAR=((''),OVR),                                     X00270000
               ERRCOUNT=((20),OVR),                                  X00280000
               ERRUNIT=((6),OVR),                                    X00283000
               FILEHIST=((ON),OVR),                                  X00286000
               HEAP=((2032,4080,ANYWHERE,KEEP,2032,2032),OVR),       X00290000
               HEAPCHK=((OFF,1,0),OVR),                              X00292000
               HEAPPOOLS=((OFF,8,10,32,10,128,10,256,10,1024,10,2048, X00293000
               10),OVR),                                             X00294000
               INQPCOPN=((ON),OVR),                                  X00295000
               INTERRUPT=((OFF),OVR),                                X00300000
               LIBRARY=((SYSCEE),OVR),                               X00305000
               LIBSTACK=((496,496,FREE),OVR),                        X00310000
               MSGFILE=((SYSOUT,FBA,121,0,NOENQ),OVR),               X00324990
               MSGQ=((15),OVR),                                      X00330000
               NATLANG=((ENU),OVR),                                  X00340000
               NOAUTOTASK=(OVR),                                     X00343000
               NONONIPTSTACK=((100,2032,ANY,KEEP),OVR),              X00347000
               NOTEST=((ALL,*,PROMPT,INSPPREF),OVR),                 X00350000
               NOUSRHDLR=((),OVR),                                   X00351000
               OCSTATUS=((ON),OVR),                                  X00353000
               PC=((OFF),OVR),                                       X00355000
               PLITASKCOUNT=((20),OVR),                              X00357000
               POSIX=((OFF),OVR),                                    X00360000
               PROFILE=((OFF,''),OVR),                               X00361000
               PRTUNIT=((6),OVR),                                    X00362000
               PUNUNIT=((7),OVR),                                    X00364000
```

```
               RDRUNIT=((5),OVR),                                          X00366000
               RECPAD=((OFF),OVR),                                         X00368000
               RPTOPTS=((OFF),OVR),                                        X00370000
               RPTSTG=((OFF),OVR),                                         X00380000
               RTEREUS=((OFF),OVR),                                        X00390000
               RTLS=((OFF),OVR),                                           X00395000
               SIMVRD=((OFF),OVR),                                         X00400000
               STACK=((2032,8176,ANY,KEEP),OVR),                           X00410000
               STORAGE=((NONE,NONE,NONE,0),OVR),                           X00420000
               TERMTHDACT=((QUIET),OVR),                                   X00430000
               THREADHEAP=((0100,0100,ANYWHERE,KEEP),OVR),                 X00004350
               TRACE=((OFF,00,DUMP,LE=0),OVR),                             X00440000
               TRAP=((ON),OVR),                                            X00450000
               UPSI=((00000000),OVR),                                      X00460000
               VCTRSAVE=((OFF),OVR),                                       X00470000
               VERSION=((''),OVR),                                         X00475000
               XUFLOW=((AUTO),OVR)                                          00480000
         END                                                                00510000
/*
//*********************************************************************
//************  LINK EDIT THE OBJECT DECK     *************************
//*********************************************************************
//LINK     EXEC PGM=IEWL,
//           PARM='LET,LIST,XREF,MAP,XREF,RENT,REUS,REFR'
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSLMOD  DD DISP=SHR,DSN=SITE.SPECIAL.CDMSLIB
//SYSLIB   DD DISP=SHR,DSN=CEE.SCEERUN          **IBM CEE LOADLIB
//OBJECT   DD DISP=(OLD,DELETE),DSN=&&OBJECT
//SYSLIN   DD *
 INCLUDE OBJECT
 INCLUDE SYSLIB(CEEPIPI)
 ENTRY CEEPIPI
 NAME CEEPIPI(R)
 INCLUDE OBJECT
 INCLUDE SYSLIB(CEEBINIT)
 ENTRY CEEBINIT
 NAME CEEBINIT(R)
/*
//
```

Except as discussed below, the IBM-supplied default run-time options can be used with any site-specific desired modifications. Note that the MSGFILE parameter is ignored and messages are sent to the CA-IDMS log file.

Recommended settings for certain parameters are as shown below. For more details on these parameters see the *IBM Language Environment for OS/390 Customization* manual.

- **ABTERMENC=(RETCODE) or ABTERMENC=(ABEND)**

  This parameter affects the action taken when an LE enclave ends with an unhandled condition of severity 2 or higher. If RETCODE code is specified, the DC task will abend with message DC128004. If ABEND is specified, the DC task will abend with a U*xxx* where *xxx* corresponds to the hexidecimal value of the user abend code set by LE. For example, an LE user abend 4093 would result in a DC task abend with code UFFD.

- **ALL31=(ON)**

  This parameter will minimize the amount of below-the-line storage, which will be allocated by LE. This parameter requires that no COBOL programs are compiled with compiler option DATA(24) and that no programs which will utilize the run-time LE are linked AMODE(24).

- **INTERRUPT=(OFF)**

  Attention interrupts are handled by the CA-IDMS/DC system and not by LE run-time support. Application PL/I programs can test for attention interrupts using the DC-ATTN-INT condition name under LE just as with earlier PL/I run-time environments.

- **POSIX=(OFF)**

  POSIX is not supported under CA-IDMS/DC/UCF.

- **RPTSTG=(OFF) or RPTSTG=(ON)**

  Normally OFF should be specified. OFF must be specified for systems prior to Release 14.1.

  The purpose of RPTSTG is to determine the storage utilization for a particular application. The report is produced at the end of a LE process and is written to the CA-IDMS log file. For efficiency reasons, the termination phase of LE processing is normally not executed in an online DC environment. If it is necessary to obtain storage information for a particular application, optional bit 196 can be set (see Appendix M in the *CA-IDMS DML Reference - COBOL* manual). Note that this option adversely affects performance. Storage reports are therefore normally produced only in a test or development system.

- **TERMTHDACT=(QUIET) or TERMTHDACT=(TRACE)**

  This option controls the extent of LE run-time information which will be supplied when an application terminates. All messages will be written to the DC log file.

- **TRAP=(ON) or TRAP=(OFF)**

  If ON is specified, program checks in an LE application will result in IBM LE error-handling being put into effect. PL/I-specific and LE messages will be written to the log. After these messages are written and the LE process ends abnormally, the DC task will abend with message DC128004 and a task snap will be taken.

  If OFF is specified, program checks in an LE application will result in an immediate task snap. This is similar to the result in a PL/I Release 2.3 run-time environment. No LE messages related to the program check will be written. Furthermore, if any PL/I applications are included in the online system, any ON ERROR clauses will not be handled properly.

In addition to the parameters above, we strongly recommend that you use smaller values than the default ones for the various heap (e.g., ANYHEAP, BELOWHEAP, HEAP) and stack (e.g., LIBSTACK, STACK) parameters since these are allocated on a task thread basis. Storage allocation is most efficient if relatively large values are specified as sixteen bytes less than a multiple of 4096. Smaller values than 4096 should be set for some parameters to avoid wasting storage. The following values have been found to be suitable for most CA-IDMS/DC/UCF systems:

```
ANYHEAP=(2032,8176,ANYWHERE,FREE)
BELOWHEAP=(496,496,FREE)
HEAP=(2032,4080,ANYWHERE,KEEP,4080,4080)
```

```
LIBSTACK=(100,2032,FREE)
NONONIPTSTACK=(4080,4080,BELOW,KEEP)
STACK=(4080,8176,ANY,KEEP)
STORAGE=(NONE,NONE,NONE,4080)
THREADHEAP=(2032,4080,,ANYWHERE,KEEP)
```

F.4 Supported LE functions

CA-IDMS/DC supports these LE functions:

- Math services
- National language support services

CA-IDMS/DC also supports storage management services, but for performance reasons, they are not recommended. The storage management services are:

- CEECRHP: Create heap segment
- CEECZST: Re-allocate (change size of) heap storage
- CEEDSHP: Discard heap segment
- CEEFRST: Free heap storage
- CEEGTST: Get heap storage

F.5 Unsupported LE functions

CA-IDMS/DC does not support the following LE functions:

- CEE3PRM: Get exec parms
- CEETDLI: Call IMS
- CEETEST: Invoke debugging environment
- Date and time services — Use the DML GET TIME command instead

Appendix G. 18-Byte Communications Blocks

G.1 Overview

As an alternative to using the 16-byte IDMS-DB and IDMS-DC communications blocks, you can specify 18-byte blocks. The difference between 16-byte blocks and 18-byte blocks is that an 18-byte block contains an additional 18-byte filler field, and the following fields are 18 bytes instead of 16 bytes:

- RECORD_NAME
- AREA_NAME
- ERROR_SET
- ERROR_RECORD
- ERROR_AREA

This appendix describes where to specify an 18-byte communications block and contains figures showing these blocks.

►► For descriptions of the fields in IDMS-DB and IDMS-DC communications blocks, see 3.2.1, “IDMS-DB communications block” on page 3-4 and 3.2.3, “IDMS-DC communications block” on page 3-12.

Where to specify the 18-byte block: For PL/I, you specify an 18-byte communications block in the SUBSCHEMA_NAMES LENGTH clause of the DECLARE SUBSCHEMA precompiler-directive statement (see 5.2, “DECLARE SUBSCHEMA” on page 5-4).

18-byte IDMS-DB block: The following figure shows the 18-byte IDMS-DB communications block.

18-byte IDMS-DB communications block

	Positions	Field	Data Type	Length (bytes)	Suggested Initial Value
*	1 8	PROGRAM_NAME	Alphanumeric	8	Program Name
	9 12	ERROR_STATUS	Alphanumeric	4	'1400'
	13 16	DBKEY	Binary	4 (Fullword)	0000
	17 34	RECORD_NAME	Alphanumeric	18	Spaces
	35 52	AREA_NAME	Alphanumeric	18	Spaces
	53 70	FILLER	Alphanumeric	18	Spaces
	71 88	ERROR_SET	Alphanumeric	18	Spaces
	89 106	ERROR_RECORD	Alphanumeric	18	Spaces
	107 124	ERROR_AREA	Alphanumeric	18	Spaces
	125 128	PAGE_INFO	Binary	4 (Fullword)	0000
	125 ... 224	IDBMSCOM_AREA	Alphanumeric	100	Spaces
	225 228	DIRECT_DBKEY	Binary	4 (Fullword)	0000
NON-CICS	229 235	Reserved for system	Alphanumeric	7	Spaces
NON-CICS	236	FILLER	...	1	...
NON-CICS	237 240	RECORD_OCCUR	Binary	4 (Fullword)	0000
NON-CICS	241 244	DML_SEQUENCE	Binary	4 (Fullword)	0000
CICS	229 244	FILLER	...	16	Spaces
CICS	245 251	Reserved for system	Alphanumeric	7	Spaces
CICS	252	FILLER	...	1	...
CICS	253 256	RECORD_OCCUR	Binary	4 (Fullword)	0000
CICS	257 260	DML_SEQUENCE	Binary	4 (Fullword)	0000

* word aligned

18-byte IDMS-DC block: The following figure shows the 18-byte IDMS-DC communications block.

18-byte IDMS-DC communications block

	Bytes	Field	Data Type	Length (bytes)	Suggested Initial Value
*	1 8	PROGRAM	Alphanumeric	8	Program Name
	9 12	ERROR_STATUS	Alphanumeric	4	'1400'
	13 16	DBKEY	Binary	4 (Fullword)	0000
	17 34	RECORD_NAME	Alphanumeric	18	Spaces
	35 52	AREA_NAME	Alphanumeric	18	Spaces
	53 70	FILLER	Alphanumeric	18	Spaces
	71 88	ERROR_SET	Alphanumeric	18	Spaces
	89 106	ERROR_RECORD	Alphanumeric	18	Spaces
	107 124	ERROR_AREA	Alphanumeric	18	Spaces
	125 128	PAGE_INFO	Binary	4 (Fullword)	0000
	125 ... 224	IDBMSCOM_AREA	Alphanumeric	100	Spaces
	225 228	DIRECT_DBKEY	Binary	4	0000
	229 ... 328	DCBMSCOM_AREA	Alphanumeric	100	Spaces
	329 332	SSC_ERRSTAT_SAVE	Alphanumeric	4	Spaces
	333 336	SSC_DMLSEQ_SAVE	Binary	4 (Fullword)	0000
	337 340	DML_SEQUENCE	Binary	4 (Fullword)	0000
	341 344	RECORD_OCCUR	Binary	4 (Fullword)	0000
	345 348	SUBSCHEMA_CTRL_END	Alphanumeric	4	Spaces

* word aligned

Index

Numerics

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

Q

R